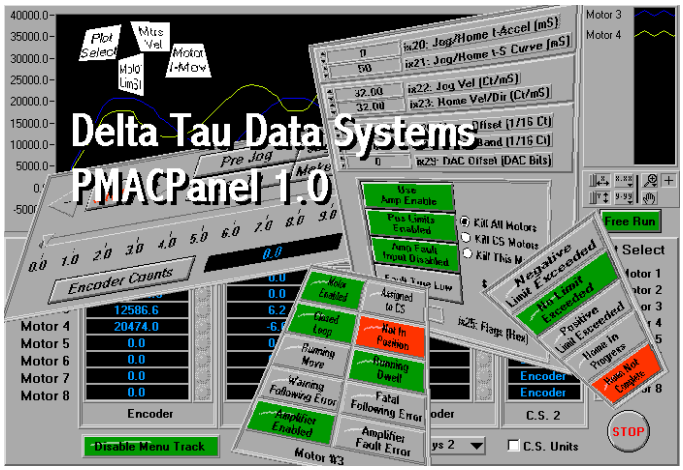
USER MANUAL

PMAC PanelPRO



PMACPanelPRO

3A0-9PLPRO-xUxx

June 15, 2004

DELTA TAU
Data Systems, Inc.
NEW IDEAS IN MOTION ...

Single Source Machine Control *Power // Flexibility // Ease of Use*
21314 Lassen Street Chatsworth, CA 91311 // Tel. (818) 998-2095 Fax. (818) 998-7807 // www.deltatau.com

Copyright Information

© 2003 Delta Tau Data Systems, Inc. All rights reserved.

This document is furnished for the customers of Delta Tau Data Systems, Inc. Other uses are unauthorized without written permission of Delta Tau Data Systems, Inc. Information contained in this manual may be updated from time-to-time due to product improvements, etc., and may not conform in every respect to former issues.

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support
Phone: (818) 717-5656
Fax: (818) 998-7807
Email: support@deltatau.com
Website: http://www.deltatau.com

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

Table of Contents

INTRODUCTION **1**
Manual Layout 1
Use Caution When Running the Examples 1
Safety Summary 1
Motion Commands *1*
Keep Away from Live Circuits *1*
Live Circuit Contact Procedures *2*
Electrostatic Sensitive Devices *2*
Hardware Interfaces *2*
Magnetic Media *2*
Technical Support 2

GETTING STARTED **3**
Unpacking and Inspection 3
PMAC Compatibility 3
Customer-Furnished Hardware 3
Customer-Furnished Software 3
Delta Tau Software *3*
National Instruments Software *4*
Microsoft Software *4*
PMAC Options for PMAC Panel Pro 4
PMAC Panel Pro and the Computer's Display 4
Installing PMAC Panel Pro 4
PMAC Panel Pro Software *4*
Configuring the Device Driver 5
Testing the Device Driver 5
Configuring LabVIEW 6
Installing the Release View *6*
Creating a Custom View *7*
Mass Compilation *7*
On-Line Help *8*
Configuring PMAC Panel Pro Communication 8
Testing PMAC Panel Pro Communication 9
PMAC Communication I-Variables *9*
PComm 32 Pro Communication Buffers *10*
Trouble Shooting PMAC Panel Pro Communication 10

PMAC PANEL PRO BASICS **11**
PMAC Panel Pro and PMAC as Client and Server 11
Application Development Components 12
Pewin 32 Pro - PMAC Executive *12*
PtalkDTPRO - ActiveX Controls for Visual C++ and Visual Basic *12*
PMAC Panel Pro - PMAC for LabVIEW 5.0 ..... *12*
PMAC Panel Pro Interface to PComm 32 Pro 13
Device Management *13*
Query/Response Communication *14*
LabVIEW and PMAC Numeric Data *14*
Download Management *15*
DPR Binary Data Buffers *15*
PMAC Panel Pro Organization 15
Device Management and Communication *16*
Query/Response Interface *16*
Indicators, Controls, and VIs - ICVs *16*
Motor ICVs *16*
Coordinate System ICVs *16*
Global ICVs *17*

Accessory ICVs *17*
Position Capture and Triggering ICVs *17*
Program Development and Encapsulation Tools *17*
Data Gathering and Graphical Tools *18*
Code Interface Nodes and Dual-Ported RAM *18*
Sample Applications *18*
Miscellaneous Utilities *19*
Documentation *19*

APPLICATION BASICS **21**
Basics 21
LabVIEW Techniques for PMAC Panel Pro 21
Dataflow and Sequencing *21*
Dataflow and Recurring Execution *21*
Giving up Control *21*
Execution Speed *22*
VI Reentrancy *22*
Persistent VI State *22*
Mechanical Action *22*
PMAC Panel Pro Indicator and Control Clusters *23*
Accessing PMAC Panel Pro VIs *24*
Clusters with an Associated Function VI *25*
PMAC Panel Pro VI Terminal Conventions *26*
PMAC Panel Pro Tutorials 26
PMAC Panel Pro Communication Tutorial 27
PmacTutor1 – Accessing PComm 32 Pro *27*
PmacTutor2 – Sending Commands and Getting Responses *28*
PmacTutor2a – Communication Logging *31*
PmacTutor3 – Sending Commands Using Buttons *33*
PmacTutor4 – Button and Response VIs *34*
PmacTutor5 – Accessing PMAC Status *36*
PmacTutor6 – Accessing PMAC I-Variables *38*
PmacTutor6b – Accessing PMAC Memory *41*
PMAC Panel Pro ICVs 43
On-line Commands *44*
PmacMotor ICVs 45
PmacTutor7 – Position, Velocity, Error, and Jogging *45*
PmacTutor8 – Motor Control with Status Monitoring *49*
PmacTutor9 - Motor I-Variable Configuration *52*
PmacMotors ICVs 55
PmacTutor10 - Requesting and Plotting Motor Motion *55*
PmacGlobal ICVs 58
PmacTutor11 – Configuring PMAC's Global State *58*
PmacCoord ICVs 61
PmacTutor12 – Using Coordinate System Definitions *61*
PmacTutor13 – Configuring and Monitoring Coordinate Systems *64*
PmacAcc ICVs 66
PmacTutor14 – Machine Input and Output *66*
PmacTutor15 – ACC16D Control Panel *67*

DEVELOPMENT TOOLS **69**
Tool Menus 69
Modifying the Menu *70*
Modifying PmacTerminalMenu *71*
Basic Tool VI Requirements *71*
Basic Tool VI Configuration *72*
PmacTerminal 73
Basic Terminal 101 ..... *73*

Basic Command Editing *73*
Buffer Management *74*
Terminal Indicators *74*
Terminal Controls *74*
Implementation Diagram *75*
PmacTerminalJog 77
PmacTerminalEdit 78
Encapsulating Motion Programs *79*
PmacTerminalExecute 80
PmacTerminalMotors 82
PmacTerminalMotorX-Y 84
PmacTerminalGather 85
Specifying Gather Addresses *88*

ENCAPSULATED MOTION PROGRAMS AND PQMS 91
PmacProgSubVI 91
PmacPQMExamp 93
PmacPQM Clusters *95*
PmacPQM Conversions *96*
PmacPQM Datalogging *97*
Using Encapsulated Motion Programs 98
PmacTestExamp 98

HOMING, ENCODERS, AND POSITION CAPTURE 103
Position Basics 103
Position-Capture 104
Trigger Condition *104*
Homing 104
Action on Trigger *104*
Home Complete *105*
Home Position Offset *105*
Zero-Move Homing *105*
Homing into a Limit Switch *106*
Homing from PLC and Motion Programs *106*
PmacHomeExamp 106
Configuring the Position Capture Trigger *107*
Monitoring the Home Position Capture *109*
Home Position Transformations *111*
Encapsulated PLC Programs 113

ENCODER CAPTURE AND COMPARE OPERATION 115
PmacEncoderPositionExamp 115
Encoder Position Transformations *115*
Position-Capture for Non-Homing Purposes 117
PLC Capture Flag Processing *117*
PmacEncoderCaptureExamp 117
External Triggers for Position Capture *119*
PMAC Position Compare Operation 119
Required M-Variables *120*
Pre-loading the Compare Position *120*
Triggering External Action *120*
PLC Compare Handling *120*
PmacEncoderCompareExamp 121
Method 1 – PLC Operation *124*
Method 2 – One-Shot Operation *124*
Method 3 – PMAC Panel Pro Interval Generation *124*
PmacEncoder Registers 125
Encoder Register Access *125*

PMAC AND NI-DAQ INTERFACING 127
External PMAC Signals 127
Compare-Equals Outputs (JEQU) 127
Servo Clock (JRS232) 128
General Purpose Digital Inputs and Outputs 128
Synchronous M-Variables 129
Position Capture FLAGs 130
DAQ Signals 131
Analog I/O Channels 131
Trigger and Scan Clock Connections 131
PmacDAQMove 131
PMAC and AT MI0-16 Signal Connections 132
Single Trigger DAQ 134
Multi-Trigger DAQ 134
Multi-Trigger DAQ with Servo Clock Sampling 135
Further Sampling Options 136
Other Interface Options 136

PCOMM 32 PRO CODE INTERFACE NODES 137
LabVIEW Code Interface Node Basics 137
What is a CIN? 137
Setting up a PMAC Panel Pro CIN Configuration 137
Adding PComm 32 Pro Include Path 137
Adding Pmac.lib to Project 138
Configuring the IDE 138
The Easy Way to Add New Projects 138
Multiple CIN Projects in a Workspace 138
Creating a CIN C-Stub for PComm 32 Pro 139

DPR – DUAL-PORTED RAM 141
Required Background Understanding 141
General Architecture Notes 142
PmacDPRRealTime 142
PmacDPRRealTimeExample 142
PmacDPRRealTimeVectorExample 148
PmacDPRFixedBack 154
PmacDPRFixedBackExample 154
PmacDPRNumeric 157
DPR Addresses and Data Organization 157
PmacDPRNumericExample 157
PmacDPRNumericClusterExample 161
PmacDPRNumericCINClusterExample 162
PmacDPRNumericSlaveExample 164
PmacDPRVarBack 166
PmacDPRVarBackExample 166
PmacDPRVarBackVectorExample 169

INTERRUPTS 171
PmacInterruptExamp 171

INTRODUCTION

When PMAC is selected for the motion control portion of a DAQ application, an integrated precision motion programming system with incredible capabilities is available.

With PMAC Panel Pro, an entirely new world of motion control applications and capabilities opens. Motion that triggers acquisitions and responds to data gathered by SCXI, VXI, and industrial automation networks such as DeviceNet and Field Bus is now possible using LabVIEW's popular and powerful graphical programming environment.

PMAC Panel Pro is an easily extensible set of more than 250 Virtual Instruments (VIs), Indicators, and Controls that communicates with and controls PMAC from LabVIEW. It allows creation of a LabVIEW application that can monitor and control everything PMAC is doing using LabVIEW while at the same time preserving an understanding of the existing PMAC interface.

Manual Layout

This manual explains how to install and use PMAC Panel Pro to develop custom applications. It assumes that the system integrator and PMAC Panel Pro developer has a basic understanding of the PMAC motion control board and LabVIEW. It does not cover the hardware and electrical configuration of PMAC or the use of Pewin 32 Pro. If questions about a particular aspect of the installation arise, do not attempt the task until a thorough understanding is gained. Contact Technical support at any time during the installation. Refer to the Technical Support paragraph below for information.

Use Caution When Running the Examples

PMAC Panel Pro has many examples to introduce itself and verify that things are working properly. Review the following issues before actually running the examples:

- PMAC Panel Pro will cause PMAC to execute motion.
- PMAC Panel Pro may require some changes in PMAC's I-Variable configuration. An I-Variable may inadvertently be changed during the execution of some of the examples. If there is a system working currently, use Pewin 32 Pro to save the configuration before making changes to the PMAC.

PMAC Panel Pro will download programs and PLCs when some of its components run. If there are motion programs and PLCs on the system currently, use Pewin 32 Pro to save them before executing those examples that utilize encapsulated motion programs and PLCs. Otherwise, they will be replaced.

Safety Summary

The following are general safety precautions not related to any specific procedures and therefore may not appear elsewhere in this publication. These are recommended precautions that all personnel using PMAC must understand and apply during different phases of operation and maintenance.

Motion Commands

Caution:

Until proper hardware safeties have been installed, configured, and tested, exercise extreme caution when moving motors to prevent damage and possible injury.

Keep Away from Live Circuits

Do not replace components or make adjustments inside equipment with power applied. Under certain conditions, dangerous potentials may exist when power has been turned off due to charges retained by capacitors. To avoid casualties, always remove power and discharge, and ground a circuit before touching it.

Live Circuit Contact Procedures

Never attempt to remove a person from a live circuit with bare hands. To do so is to risk sure and sudden death. If a person is connected to a live circuit, the following steps should be taken:

- Call for help immediately.
- De-energize the circuit, if possible.
- Use a wood or fiberglass hot stick to pull the person free of the circuit.
- Apply cardiopulmonary resuscitation (CPR) if the person has stopped breathing or is in cardiac arrest.
- Obtain immediate medical assistance.

Electrostatic Sensitive Devices

Various circuit card assemblies and electronic components may be classified as Electrostatic Discharge (ESD) sensitive devices. Equipment manufacturers recommend handling all such components in accordance with standard ESD procedures. Failure to do so may void the warranty.

Hardware Interfaces

When interfacing PMAC signals with any other data acquisition equipment, use extreme caution to avoid shorting signals to supply or ground potentials. Furthermore, observe all signal load, voltage, and current limitations. Failure to do so may void the warranty.

Magnetic Media

Motors and amplifiers may generate strong magnetic fields. Do not place or store magnetic media (tapes, discs, etc.) within ten feet of any magnetic field.

Technical Support

Delta Tau will respond to any questions or concerns regarding PMAC Panel Pro. Contact the Delta Tau Technical Support Staff using the following methods:

By Telephone

For immediate service, contact the Delta Tau Technical Support Staff by telephone Monday through Friday. The support line hours and telephone numbers are listed in the table at the end of this section.

By Fax and E-Mail

Fax or E-Mail a request or problem overnight and it will be dealt with the following business day. Fax numbers and E-Mail addresses are listed below. Supply all pertinent equipment set-up information.

World Wide Web (WWW)

Delta Tau maintains a complete website containing many manuals, product updates, help files, application notes, and programming examples. Go to www.deltatau.com.

Bulletin Board Service (BBS)

Messages may be left on one of the Bulletin Board Services (BBS). The BBS is provided for customers, distributors, representatives, and integrators, et al. The service may be downloaded and files uploaded and posted bulletins and Delta Tau newsletters read. Messages may be left for anyone who is a member/user of the Bulletin Board Systems. A modem and ProComm-Plus or similar communications program is required to use this system. Many Download-Upload Protocols such as Z-Modem are supported.

Refer to the website at www.deltatau.com for specific information.

GETTING STARTED

Unpacking and Inspection

After receiving and opening the PMAC Panel Pro package, compare the contents to the packing list to ensure everything has been received. If anything shown on the packing list is missing, contact Delta Tau immediately. Carefully inspect all components for signs of physical damage.

PMAC Panel Pro consists of:

- PMAC Panel Pro CD containing Vis and on-line documentation
- PMAC Panel Pro technical documentation package

PMAC Compatibility

PMAC Panel Pro works with the following motion control boards:

- PMAC- ISA/PCI up to eight axis
- PMAC- LITE ISA/PCI four axis
- PMAC2 – ISA/PCI four or eight axis
- Mini PMAC ISA/PCI
- Turbo PMAC/PMAC2 up to eight 8 axis
- UMAC/QMAC up to eight axis
- PC104/Turbo PC104 four or eight axis

A system integrator or other person may have performed installation and configuration of PMAC, amplifiers, and motors. Refer to the documentation provided by the integrator or with the purchase for details.

PMAC Panel Pro supports PMAC2 with the exception of certain encoder specific capabilities such as encoder capture and compare.

Customer-Furnished Hardware

In order for the PMAC Panel Pro to operate, the following customer-furnished hardware is required:

- IBM or 100% compatible 486/66 MHz personal computer (PC). Pentium® or equivalent recommended.
- Minimum of 16MBof RAM. 32MB recommended
- A minimum of 100MB of free hard disk space.
- SVGA color monitor with minimum 1024x768 resolution.
- In addition, the following optional National Instruments or third party data acquisition equipment may exist:
- Multi-function data acquisition I/O cards
- Signal conditioning equipment
- Image Acquisition
- GPIB Instrument Control
- Industrial Communication

Customer-Furnished Software

PMAC Panel Pro requires Microsoft Windows 95 (or later version) or Windows NT 4.0 to operate.

Delta Tau Software

PMAC Panel Pro requires the existence of Delta Tau's PMAC device drivers. Several possible options have already been installed or purchased with PMAC Panel Pro. The following information will help determine the steps needed to successfully install and configure PMAC Panel Pro.

- If using PMAC Executive for Windows, make sure that Pewin 32 Pro is installed and tested. Skip those installation steps involved with the installation of the PComm 32 Pro device driver.
- If PComm 32 Pro is installed and tested, skip the steps for the installation of the PComm 32 Pro device driver.

- If Ptalk DT Pro has been purchased and installed, skip the steps for the installation of the PComm 32 Pro device driver.
- If none of the above options has been purchased, install a limited edition of the driver, configure it, and test it. Certain PMAC Panel Pro capabilities may not be supported without PComm 32 Pro.

National Instruments Software

PMAC Panel Pro was developed for LabVIEW 5.0. Previous versions of LabVIEW are not supported by Delta Tau's PMAC Panel Pro motion package. LabVIEW must be installed prior to installing PMAC Panel Pro. LabVIEW patch 5.0fix2, available from National Instruments via FTP is recommended.

There may be other software from National Instruments or Third Party sources such as NI-DAQ. There are no known conflicts with PMAC Panel Pro when using these packages.

Microsoft Software

Certain PMAC Panel Pro capabilities are implemented with compiled C code. These operate perfectly as is. To modify them to suit different requirements or add other CINs for specific reasons, use Microsoft Visual C++ 5.0 or Microsoft Visual C++ 6.0. PMAC Panel Pro does not support other compilers directly.

PMAC Options for PMAC Panel Pro

PMAC Panel Pro supports a wide range of PMAC's capabilities. Some of these require the purchase of additional Delta Tau hardware and software accessories. These options include:

- PMAC's dual-ported RAM – This is required to utilize the PmacDPR collection of VIs.
- PComm 32 Pro – Complete PMAC device driver. PMAC Panel Pro provides a version of the complete driver with reduced capabilities. Tailoring of some PMAC Panel Pro capabilities may require the complete PComm 32 Pro release.
- Various PMAC I/O and accessory options

PMAC Panel Pro and the Computer's Display

PMAC Panel Pro's indicators and controls are configured for display on a computer with 1024x768 resolution or greater. Set the display's resolution to at least this size to use them. PMAC Panel Pro VIs work at smaller display sizes but the panels will not fit within the display area. Either resize the panel controls or change the size of the display.

Installing PMAC Panel Pro

Because of the number of software drivers and steps in the communication process, it is important that each step be done carefully and tested before proceeding to the next. The following steps must be taken before installing PMAC Panel Pro:

- Install, configure, and test LabVIEW 5.0 or greater and any patches.
- Install, configure, and test any National Instruments boards.
- Install, configure, and test the system's PMAC hardware.
- Install, configure, and test PComm 32 Pro, Pewin 32 Pro, and/or Ptalk DT Pro if purchased. If these options were not purchased, install a limited edition of the PComm 32 Pro device driver included with PMAC Panel Pro.

PMAC Panel Pro Software

Before installing PMAC Panel Pro, read the license agreement included in this manual (behind title page). Check the release notes included with the manual and located in the Documentation directory on the CD for last minute changes. Installation of PMAC Panel Pro is done in two steps. First, the drivers must be installed. Next, the PMAC Panel Pro software must be installed.

Installation of the Driver

Skip this step if Pewin 32 Pro, PComm 32 Pro, or Ptalk DT Pro has been installed and tested. If one of these tools has not been purchased, locate the PMAC Panel Pro Drivers directory on the CD and run **Setup**. The installation program will suggest a directory path where the program files should be copied. The suggested directory location is c:\Program Files\Delta Tau\PMAC Panel Pro. This will install the drivers and the test application PMAC Test Pro.

Installation of PMAC Panel Pro

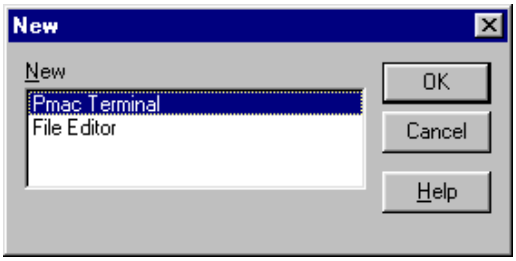
To install PMAC Panel Pro, locate the PMAC Panel Pro directory on the CD and run **Setup**. If LabVIEW has been installed properly, the installation program will add several components to the LabVIEW installation. If it cannot locate LabVIEW, specify its location or exit the installation and install LabVIEW. The primary PMAC Panel Pro component is the PMAC Panel Pro.lib directory in the LabVIEW installation directory. The library directory contains numerous sub-directories to organize the VIs, utilities, and documentation.

Configuring the Device Driver

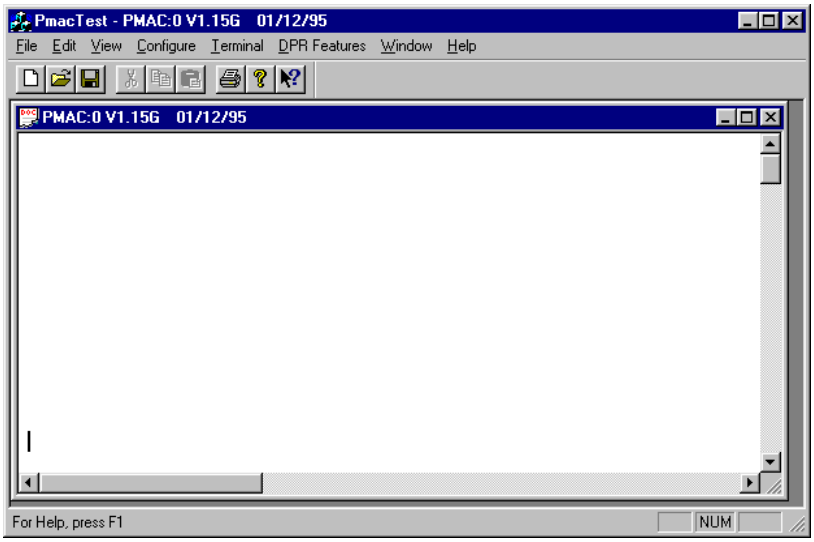
Refer to PComm 32 Pro_Install.pdf for Device driver installation and configuration for all supported operating systems and modes of communication.

Testing the Device Driver

Initial testing of PMAC and the device driver is accomplished with the program PMAC Test Pro included with PMAC Panel Pro or the PComm 32 Pro drivers. When PMACTestPRO executes, the following dialog appears requesting the preferred operational mode.



Click **OK** and a terminal window will appear.


PmacTest - PMAC:0 V1.15G 01/12/95
File Edit View Configure Terminal DPR Features Window Help
PMAC:0 V1.15G 01/12/95
For Help, press F1
NUM

Caution:

Until the proper hardware safeties have been installed, configured, and tested, use caution when moving motors to prevent damage and possible injury.

PMAC Test Pro is now in terminal emulation mode, allowing interaction directly with PMAC. Although it is tempting to move motors when communication is first established in this step be thoroughly familiar with the mechanical setup and be certain that commands executed from PMAC Test Pro will not cause damage or injury.

Check to see if there is a response by typing **I10<Enter>**. PMAC should respond with a six or seven digit number. Now type **III<Enter>**. PMAC should respond with a beep, signifying an unrecognized command. Next, type **P<Enter>**. This requests a position. PMAC should respond with a number.

Note:

If the system contains a PMAC Lite, the response will still contain eight numbers when **<CONTROL-F>** is entered.

Now type **<CONTROL-F>**. The system should respond with eight numbers (one for each axis) since **<CONTROL-F>** requests following errors from all eight motors (some or all may be zero). Note that even with encoder counts as read-out (no scaling), PMAC's position is displayed with fractional counts.

If error dialogs appear or the responses are not as specified, check PMAC Test Pro's help capability. It might help to review the Configuring the Driver section. If the problem persists, contact Technical Support.

The PMAC and the PComm 32 Pro communication drivers are now installed on the system.

Configuring LabVIEW

During the installation of PMAC Panel Pro, the directory PMAC Panel Pro.lib containing the PMAC Panel Pro distribution VIs was created in the LabVIEW directory. Three things need to be done to seamlessly integrate PMAC Panel Pro into the LabVIEW development environment:

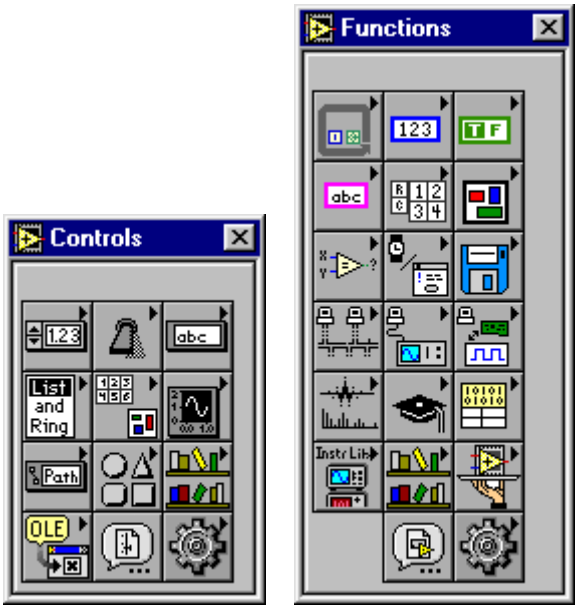
1. Create a view.
2. Mass compile the Vis.
3. Configure PMAC Panel Pro for the PMAC driver configuration.

Installing the Release View

To facilitate the use of PMAC Panel Pro, install the view contained in the release or create a custom view so that the PMAC Panel Pro VIs and controls are easily accessible from the Controls and Functions palettes and do not clutter the User.lib directory. The procedure for doing this is outlined here:

1. Run LabVIEW
2. Select **Edit»Select Palette Set»PMAC Panel Pro**. This view is a modification of the default view. If no PMAC Panel Pro selection appears, check for a directory named PMAC Panel Pro in the directory LabVIEW\Menus.

The Controls and Functions palettes will appear as:



Access to the entire suite of PMAC Panel Pro controls and function VIs is now available using the

icon and its sub-palettes. PMAC Panel Pro icons are text based and indicate the PMAC Panel Pro.lib subdirectory they are located in and their specific function.

Creating a Custom View

If PMAC Panel Pro in PMAC Panel Pro.lib have not been installed or there is already a custom view to accommodate other LabVIEW packages, refer to the LabVIEW manuals or Online Reference under the topic Palettes Editor to add PMAC Panel Pro to the palettes. Use the **Edit»Edit Control and Function Palettes…** option.

1. Create a new view.
2. Insert a Submenu and link it to a directory - PMAC Panel Pro.lib (or a custom name).

The icon **\PmacDocument\PMAC Panel ProIcon.bmp** can be added during the palette editing process.

Mass Compilation

This step compiles the entire PMAC Panel Pro release so that there are fewer searches when loading VIs and confirms that everything can be found. Select **Edit»Mass Compile** to display the file selection dialog while there is a VI open. Browse to the PMAC Panel Pro.lib directory and click **Select Cur Dir**. LabVIEW will then begin loading and compiling the entire PMAC Panel Pro release. When this is complete click **Cancel**.

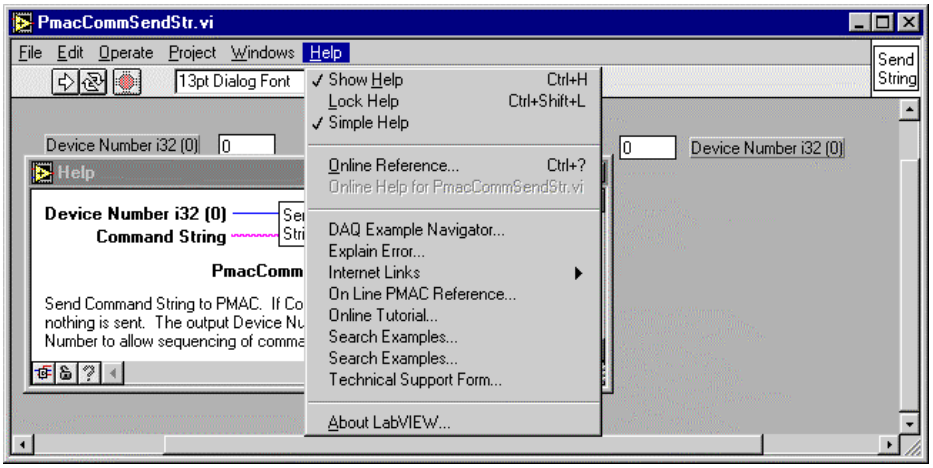
Note:

If the PComm 32 Pro package has been purchased, the ability to develop LabVIEW Code Interface Nodes is available that may require re-compilation. See the PComm 32 Pro Code Interface Nodes section for details.

If the compilation process encounters problems, note the error message. The most common problem encountered will be its inability to locate the PComm 32 Pro driver extension Pmac.dll installed by the Pewin 32 Pro, PComm 32 Pro, PTalk DT Pro, or PMAC Panel Pro. This should be located in c:\Windows\System. If the file is not in this directory, try to locate it and determine where the file was placed. The file can be copied into the correct directory and reattempt the compilation.

On-Line Help

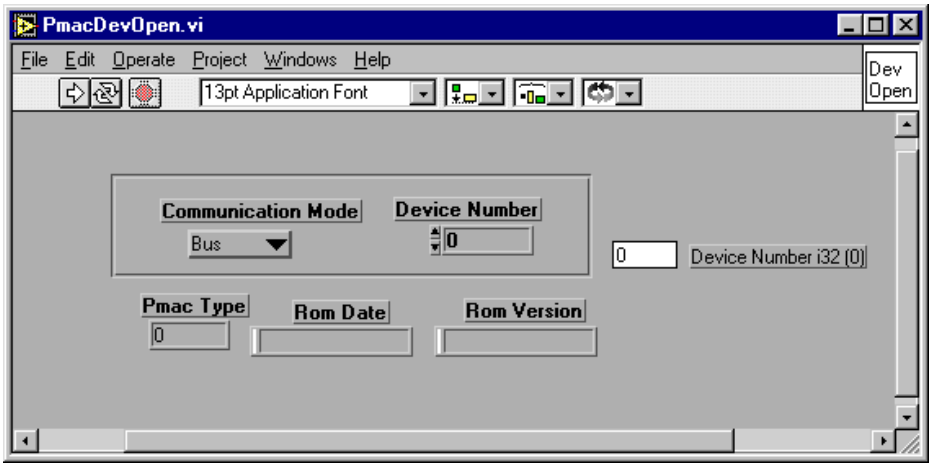
PMAC Panel Pro has extensive on-line help in two forms. Extensive documentation of every VI and its terminals is available using the standard LabVIEW **Help»Show Help** option. There are standard Windows on-line help files located in the LabVIEW\Help sub-directory. These include help versions of the printed manuals and several PMAC manuals. This is useful when examining the examples and tutorials. When LabVIEW starts, the contents of this directory are parsed and any help files located in the directory are added to LabVIEW's pull down help menu. The figure below shows both forms of help. Note the entry in the pull down menu for On Line PMAC Reference. Other on-line help files are added as needed. Many of these can be down loaded from Delta Tau's web site and copied into the LabVIEW\Help directory.



Configuring PMAC Panel Pro Communication

PMAC Panel Pro communicates with PMAC using the PComm 32 Pro device driver configured previously. To access the driver from PMAC Panel Pro, define the device number and communication mode for the driver for PMAC Panel Pro. PMAC Panel Pro's primary device driver, PmacDevOpen, is configured using the following procedure

1. Run **LabVIEW.**
2. Open the PmacDevOpen VI. Select **File»Open** and navigate to PMAC Panel Pro.lib\PmacDevice\ PmacDevOpen.vi. The following panel should appear:

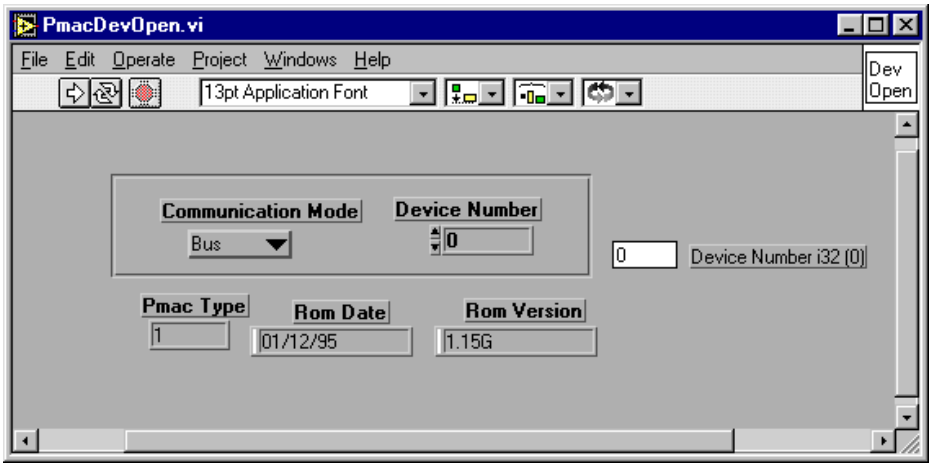


Set the Device Number control - not the indicator in the white box - on the front panel to the device number specified for the PMAC during the configuration of the driver. The default Device Number in a new PMAC Panel Pro package is 0. If this is the device, no change is required. Otherwise, modify the control and make the value permanent using the right mouse button and clicking the **Data Operations»Make Current Value Default** option, and then saving the VI.

The device driver manager allows the selection of Serial Port or In PC Bus as the desired communication mode. The same selection should appear in the Communication Mode drop list on the front panel. The default mode specifies the use of the Bus. If this is the correct mode, no change is required. If the desired communication mode is DPR, the device driver control panel should specify In PC Bus along with a valid DPR address. The Communication Mode in this panel should display DPR. As with the device number, make this value permanent by using the right mouse button, clicking the **Data Operations»Make Current Value Default** option, and then saving the VI.

Testing PMAC Panel Pro Communication

The final step in the installation of PMAC Panel Pro is to test its ability to communicate with the device driver. After configuring PmacDevOpen and saving the default changes, execute it using the run button on the menu bar. The panel should change to reflect the Type, Rom Date, and Rom Version of the PMAC as shown here.



The PMAC and the PComm 32 Pro communication drivers have been installed on the system.

PMAC Communication I-Variables

PComm 32 Pro supports communication with PMAC using UNICODE and standard C/C++ ASCII strings. PMAC Panel Pro is configured to use C/C++ ASCII strings not UNICODE due to the use of LabVIEW's Call Library VI to interface LabVIEW with PComm 32 Pro.

PMAC uses I-Variables to define communication characteristics. These are important to verify because the behavior of PMAC Panel Pro depends on their value. If PMAC Panel Pro communication is not operating properly, configure these values using Pewin 32 Pro or the PMACTest Pro application included with PMAC Panel Pro.

I1 - Serial Port Mode

This parameter specifies whether PMAC will use hardware flow control using CS and whether a software card address is required with each command. At present no software card address is required hence I1 should have a value of 0 or 1. If desired, a card address can be pre-pended to all communication with some modifications to PmacCommSendStr, PmacCommGetStr, and PmacCommRespStr.

I3 - I/O Handshake Control

This parameter determines what characters, if any PMAC uses to delimit a transmitted line, and whether PMAC issues an acknowledgment (handshake) in response to a command. The preferred setting is I3 = 2.

I4 - Communication Integrity Mode

This parameter allows PMAC to compute checksums for the communication bytes sent between PComm 32 Pro and PMAC. This value should be I4 = 0 because PMAC Panel Pro does not currently add or strip a checksum.

I6 - Error Reporting Mode

This parameter specifies how PMAC reports command line errors. The preferred setting is I6 = 1. In this mode, PMAC errors are properly parsed by PMAC Panel Pro and reported to the user with a pop-up dialog.

I58 - DPRAM ASCII Communication Enable

This parameter enables or disables the DPRAM ASCII communications function. When I58 = 1 PMAC Panel Pro sends and receives communication through DPRAM. When I58 = 0 communication is done via the Bus or Serial Port. Enabling ASCII communication is not required to access DPRAM available using other PMAC Panel Pro capabilities. Using DPRAM ASCII communication modifies I3 and will disable PMAC Panel Pro's ability to properly parse error messages effectively. The preferred value is I58 = 0. This implies that Bus or Serial is the preferred Communication Mode to be specified for PmacDevOpen. This has little impact on overall performance and does not preclude the use of DPRAM for memory mapped purposes.

PComm 32 Pro Communication Buffers

PMAC handles commands and responses in a very simple manner. PMAC commands that generate responses place the responses in an internal buffer that is transferred into the caller's buffer. If the entire response does not fit in the caller's buffer, the data is held in PMAC until the remainder of the buffer is fetched. New commands sent to PMAC flush the response buffer prior to executing the new command. Hence, responses that are not fully retrieved are lost.

Communication with PMAC via PComm 32 Pro requires empty buffers into which responses are placed. The empty response buffers for PmacCommGetStr and PmacCommRespStr are created as 128-byte buffers. If larger default buffers are needed, the size of the buffer can be increased to 256. PComm 32 Pro internals cannot handle buffers larger than this. PMAC Panel Pro handles larger response buffers internally using PmacCommGetBuffer.

Trouble Shooting PMAC Panel Pro Communication

At this point, it is assumed that the driver was successfully configured and tested as outlined in the Configuring the Device Driver and Testing the Device Driver sections. If these steps were skipped, review them now. If Pewin 32 Pro or PComm 32 Pro are installed and they work, the problem is with the configuration of PmacDevOpen or the communication I-Variables.

- In the event that LabVIEW crashes when running PmacDevOpen, reboot the computer to eliminate any damage to the driver and memory caused by the crash.
- Verify the correct operation of the device driver by checking the configuration using MotionExe or the Control Panel Applet. When **OK** is selected from the setup dialog, the driver attempts to contact PMAC and reports the success or failure of the attempt.
- Check the driver operation communication with PmacTest Pro or Pewin 32 Pro.
- Revisit the driver configuration and make certain that the device number and communication modes specified match those specified for PmacDevOpen. Make changes to the VI and retest the communication by running the VI again.

Early versions of Windows 95 do not work well with LabVIEW and the PComm 32 Pro device driver. If the problem persists, contact Technical Support. If the system continues to crash, try to note any error messages in detail.

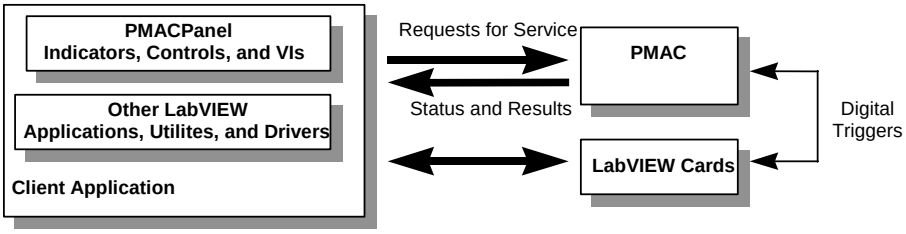
PMAC PANEL PRO BASICS

PMAC Panel Pro and PMAC as Client and Server

PMAC Panel Pro is a powerful LabVIEW tool kit that allows the development of GUI-based clients requiring precision multi-axis motion that integrate PMAC's unique capabilities with other LabVIEW devices.

PMAC Panel Pro is not intended to replace a thorough understanding of PMAC's powerful motion and PLC capabilities, its architecture, its command language, or its programming language. PMAC Panel Pro is not a Graphical Motion Language (GML) that allows the writing of PMAC programs by stringing together a set of motion description VIs.

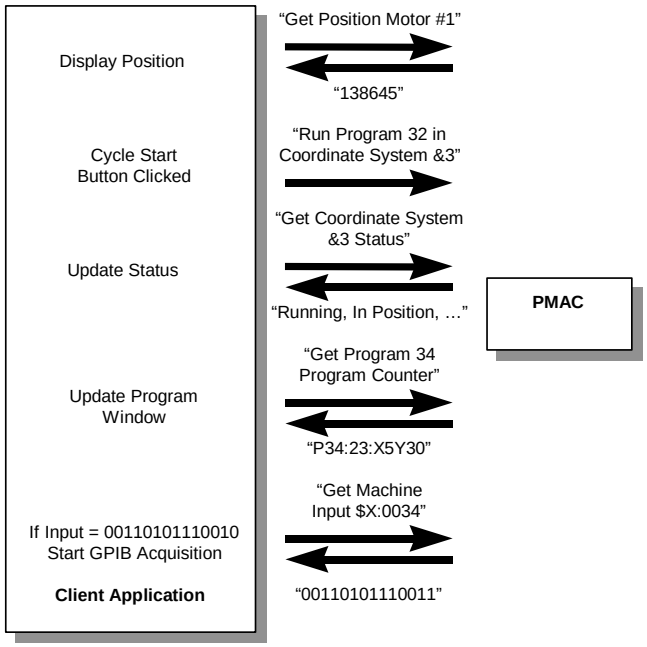
The client/server architecture, illustrated below, works exceedingly well with LabVIEW's G-Code data flow execution model and matches the model used to communicate with GPIB, Industrial Automation networks, etc. PMAC Panel Pro applications place requests to PMAC to set motion characteristics, run motion programs, and configure and monitor PMAC status. PMAC executes these requests without the assistance of the client application as long as power is applied. Using this model, PMAC Panel Pro applications can use data from other LabVIEW devices to control the motion and coordinate control of those devices with the execution of the motion.



The figure below illustrates a set of Query/Response exchanges between the PMAC Panel Pro client application and PMAC. Commands for service are sent by the application in response to

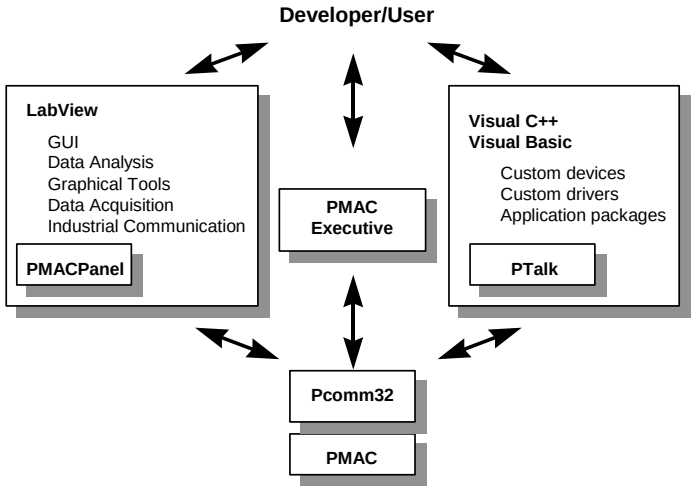
- Button clicks
- Events such as measurements made by other instruments
- Polling requirements

In response to these requests, PMAC responds with the requested operation or data. Because of this architecture, everything PMAC does is under control of and in coordination with client applications.



Application Development Components

Depending on the motion requirements, system integration requirements, and performance needs, there are three methods for developing the PMAC Panel Pro applications. The following figure shows the three primary components available from Delta Tau for the system development.



Pewin 32 Pro - PMAC Executive

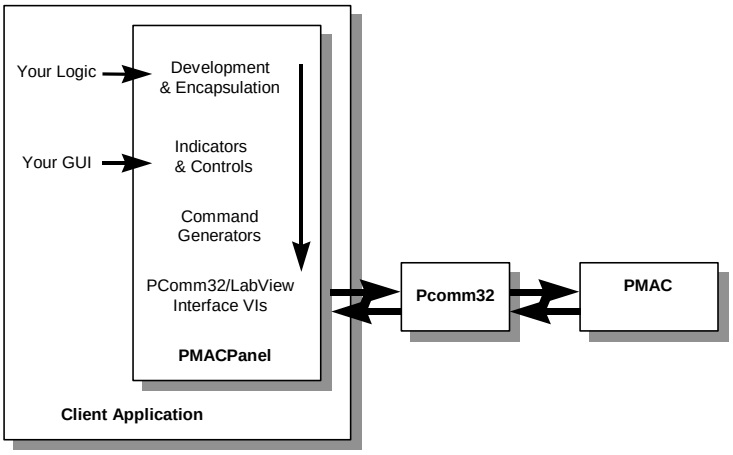
PMAC system setup and configuration operations such as motion program development, motor tuning, limit and safety configuration, etc. are best accomplished using the executive. Once a set of sample motion programs is available, a PMAC Panel Pro application can be developed. PMAC can still operate as a standalone controller. To interact with PMAC, the PMAC Panel Pro applications are available to change the operation of PMAC.

PtalkDTPRO - ActiveX Controls for Visual C++ and Visual Basic

Applications with demanding computational needs can written in C++ and their needs can be communicated to PMAC using PtalkDTPRO. As an ActiveX control, Ptalk DT Pro can be used from within PMAC Panel Pro. It is also possible to compile this code into a dynamic link library or as a LabVIEW Code Interface Node and use it from custom VIs in a PMAC Panel Pro application.

PMAC Panel Pro - PMAC for LabVIEW 5.0

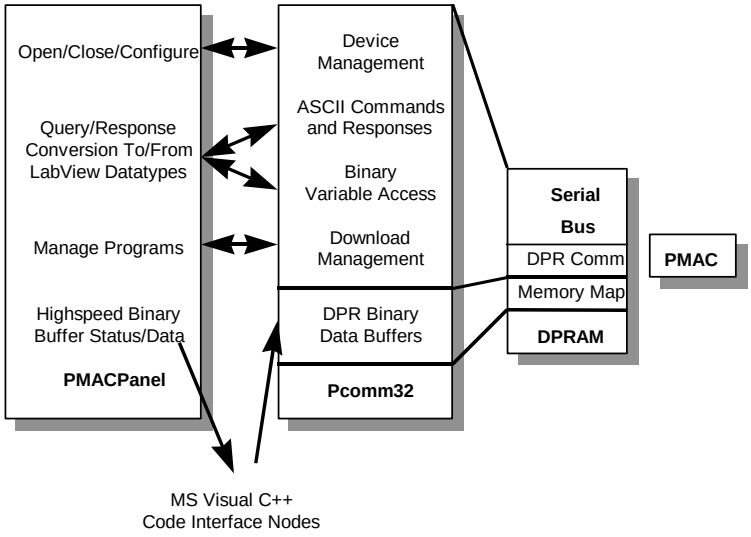
PMAC Panel Pro provides a complete suite of VIs to simplify and standardize the application's access to PMAC from LabVIEW. It is divided into four basic levels or categories. At the very lowest level is VIs that provide an interface to PComm 32 Pro. Above this are two levels that provide collections of indicators, controls, and VIs that can be used in the LabVIEW applications to control PMAC and monitor its status. Finally, a level provides program development utilities that can be used to encapsulate PMAC motion and PLC programs as VIs that are controlled by the application. In general, the top three levels are used for application development.



PMAC Panel Pro Interface to PComm 32 Pro

PMAC Panel Pro's lowest level provides a flexible set of capabilities through which all command interactions with PMAC is conducted. To do this, PMAC Panel Pro wraps selected PComm 32 Pro functions with appropriate logic to provide a transparent interface between the LabVIEW application and PMAC. It provides three primary categories of access to PMAC:

1. Device Management
2. Query/Response ASCII and Binary Communication
3. Download Management



Device Management

Access to the device driver is handled with the PmacDevOpen VI configured earlier. It defines whether Query/ Response communication uses the Serial Port, Bus, or DPR. PComm 32 Pro hides all details associated with a given mode. If the PMAC is inserted into the host computer, Bus communication is a sure bet. DPR, while slightly faster, is not required. If the PMAC is located remotely, then use the serial port to communicate with it. If the application does a lot of polling for status data, there will be a marked decrease in application execution performance.

Note:

DPR is primarily intended to pass real-time data gathering buffers and binary data for operations like inverse-kinematics between PMAC and the client. It is not required for general Query/Response interaction.

Query/Response Communication

Query/Response communication is the most basic form of communication with PMAC. This mechanism allows the application to use the entire set of PMAC's on-line Commands to interact with PMAC. The quickest way to build a PMAC Panel Pro application to control and monitor PMAC is to locate the functionality needed in the PMAC User Manual and PMAC Software Reference Manual. Test it with Pewin 32 Pro or the PmacTerminal contained in PMAC Panel Pro. Then build a LabVIEW VI and select an appropriate PMAC Panel Pro VI to send that command and/or data to PMAC.

LabVIEW and PMAC Numeric Data

LabVIEW supports a wide range of native data types that need to be communicated to PMAC. This data must be formatted for transmission to PMAC and converted from PMAC responses into LabVIEW types. PMAC returns numeric data as decimal or hexadecimal ASCII strings. The query/response VIs convert this ASCII data into native LabVIEW numbers for manipulation and display.

PMAC uses the Motorola 56K series of Digital Signal Processors for its computational engine. The memory architecture is based on a 48-bit long word comprised of two 24-bit words. This organization allows one to access the entire 48-bit long word, or either of its two 24-bit words using the same address.

The long word at memory location $23F8 is accessed using D:$23F8 or L:$23F8. The upper word is designated X:$23F8 and the lower word is designated Y:$23F8.

PMAC's firmware uses several data types depending on the quantity being represented: 1-bit booleans, 8, 16, 24, 32, and 48-bit integers, and 24, 32, and 48-bit floating point numbers. Of these, most integers are 24 or 48-bits. This presents an immediate issue that must be understood when developing a PMAC Panel Pro application. LabVIEW's short integers are 16-bits – too short for PMAC's 24-bit integers. Its long integers are 32-bits – too short for PMAC's 48-bit integers and too long for PMAC's 24 bit integers. Furthermore, LabVIEW does not support a 48-bit integer or 48-bit floating-point number. Fortunately, because Query/Response communication is conducted with ASCII strings the size of PMAC's data is somewhat hidden.

To simplify the interface between LabVIEW and PMAC, PMAC Panel Pro supports the conversion of PMAC data into one of six LabVIEW data types. Conversion into unsigned representations is done easily using LabVIEW's many conversion VIs. The following table enumerates the representations and names supported in each domain. The PMAC Panel Pro names Short, Ushort, Long, and ULong are used in the names of many Query Interface VIs.

PMAC	LabVIEW	PMAC Panel Pro
1 bit binary	Boolean	Boolean or Bool
1, 8, 16 bit integers	16 bit integer (i16/u16)	Short (i16)/UShort (u16)
16, 24, 32 bit integers	32 bit integer (i32/u32)	Long (i32)/ULong (u32)
24 bit floating point, 48 bit integers, 48 bit floating point doubles	64 bit floating point doubles	Double or Dbl

There are some situations, depending on the value of I9, where PMAC's ASCII response strings are hexadecimal not decimal. The conversion of these ASCII responses into equivalent native LabVIEW binary representations are handled by classes of VIs that use PComm 32 Pro's binary variable access capabilities.

Download Management

Maintenance of PMAC programs is provided by a collection of VIs that directly access PComm 32 Pro's download functions. These compile ASCII PMAC programs and download them to PMAC for execution.

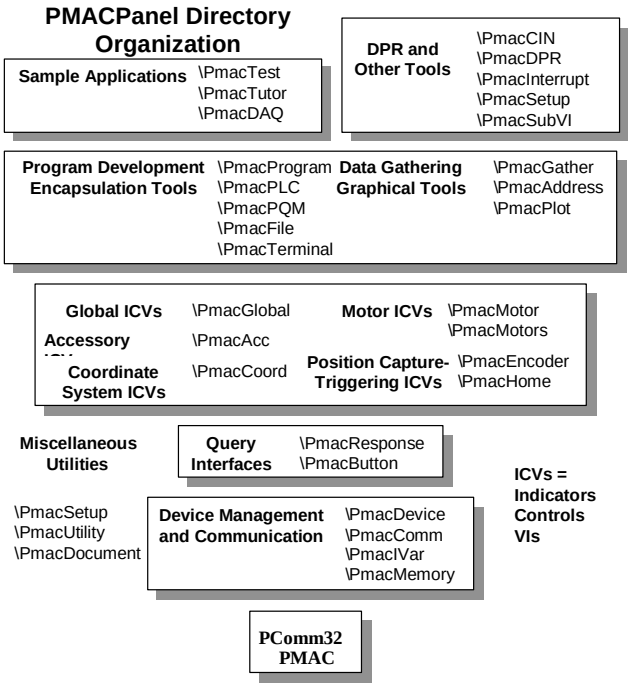
DPR Binary Data Buffers

PR represents a unique PMAC capability that rapidly transfers binary numerical data between the host and PMAC. This communication mode eliminates the string formatting and parsing required with ASCII communication. It is however, not the best solution for all problems. The DPR – Dual-Ported RAM section covers the use of these mechanisms fully.

PMAC Panel Pro Organization

This brief explanation of PMAC Panel Pro's organization will help get the information needed to quickly and painlessly, as well as help plan the application's architecture.

The PMAC Panel Pro library, contained in the directory PMAC Panel Pro.lib, is divided into five basic categories as illustrated in the following figure. These categories provide an increasing level of capability when progressing from the lower levels to the higher levels.



Within each category are several sub-directories containing collections of indicators, controls, and VIs (ICVs) that provide capabilities to make the application development task easier. The VIs in each sub-directory follow a naming convention that makes them easier to locate. For example, in the directory \PmacCoord, all VIs are named PmacCoord. The names of VIs in a given sub-directory may have shortened names to make them a little easier to fit on the screens (e.g., VIs in \PmacResponse are shortened to PmacResp). The collections of VIs in each category implement most commonly used capabilities and generally have examples to demonstrate their use. The examples will have the word Examp or Example at the end of their name. The remaining sections cover numerous tutorial exercises and examples to demonstrate their use.

Device Management and Communication

At the lowest level of the architecture is VIs that configures and manages PMAC Panel Pro communication with PMAC using PComm 32 Pro.

\PmacDevice – Configure and manage access to PMAC using PComm 32 Pro. Configuration of this access must match the driver configuration set using the control panel applet or MotionExe application.

\PmacComm – Provide the ability to send on-line ASCII command strings and data to PMAC and receive ASCII responses from PMAC. Error handling and the ability to buffer communication for analysis and future reference are provided at this level.

\PmacIVar – Provide direct binary access to PMAC I-Variables. This collection of VIs provides an easy to use architecture for accessing I-Variables and avoids formatting problems that can arise when querying PMAC I-Variables that might be returned as hexadecimal values.

\PmacMemory – Access to PMAC's memory mapped variables and registers are simplified with this group of VIs.

Query/Response Interface

PMAC's client/server interaction model works by sending requests to PMAC and awaiting responses. Interaction with PMAC at this level removes much of the tedium involved in sending commands and formatting responses.

\PmacResponse – Send formatted command strings to PMAC and convert ASCII response strings into numerical values for use in the PMAC Panel Pro application.

\PmacButton – In most user interfaces queries are sent to PMAC as the result of Boolean user actions and conditions. This group of VIs send command strings to request responses when an associated LabVIEW Boolean condition is TRUE.

Indicators, Controls, and VIs - ICVs

This collection of directories and VIs provide an extensive set of indicators, controls and VIs that will tremendously speed the development of PMAC Panel Pro applications. Each of the five groups in this category provide prepackaged indicators and controls to manipulate PMAC's I-Variable setup, monitor execution status, and send commands to PMAC using appealing clusters of LabVIEW controls. Capabilities not implemented by these VIs can be added easily by modifying the provided VIs. Numerous examples are provided to demonstrate the capabilities of the collection and can used as is.

Motor ICVs

Interaction with individual and sets of motors is provided by these collections of VIs. The collections contain VIs to request PMAC status and motor states and display their data on pre-defined cluster indicators. In addition, controls to jog and control motors are provided to simplify testing and development of programs.

\PmacMotor – Monitor and control individual motors. There are numerous ICVs to obtain motor position, velocity, and following error, modify motor I-Variables, process motor status, and jog motors.

\PmacMotors – Monitor and plot the motion of collections of motors in defined coordinate systems. Plotting tools for selecting which motors and motion variables to plot are available. Samples of real-time strip charts and XY charts are provided.

Coordinate System ICVs

\PmacCoord – Monitor the execution of programs and definition of motor coordinate systems. This information is required for the development of user interfaces in which information is entered and displayed in coordinate system units rather than encoder units.

Global ICVs

\PmacGlobal – General PMAC setup and configuration are provided by this collection for VIs. These capabilities are used for the development of supervisory VIs.

Accessory ICVs

\PmacAcc – PMAC has numerous accessories that are used to provide digital I/O, analog I/O, and control capabilities. This includes the Acc-16D front panel and the alphanumeric display. This collection provides basic ICVs that access the memory mapped data used by these accessories.

Position Capture and Triggering ICVs

PMAC has the ability to capture exact motion positions in response to external triggers and generate triggers when specified positions are reached. This capability can in conjunction with National Instruments DAQ boards, be used to capture a position or trigger instrument control in response to specified positions.

\PmacHome – This collection of VIs provides access to the I-Variables and memory variables required for defining homing criteria and determining home position offset.

\PmacEncoder – This collection provides VIs to access and control the encoder gate array for position capture and compare. There are also tools for developing background PLC programs for generating Compare-Equal outputs.

Program Development and Encapsulation Tools

Developing PLC and motion programs that work seamlessly with PMAC Panel Pro requires four major components.

1. Tools to edit and create programs
2. Tools to monitor the execution and debugging of programs
3. Tools to develop PMAC Panel Pro panels to communicate program specific data between the program and user
4. Tools to encapsulate programs, their execution, and monitoring into a single easy to use VI

The purpose of these collections is to allow the integration of motion programs into a PMAC Panel Pro application. Either Pewin 32 Pro or the PMAC Panel Pro application tools documented in the Development Tools section of this manual can be used for the development of the raw motion programs.

\PmacTerminal – Terminal application tools for interactively creating, controlling, and monitoring PMAC and the programs.

\PmacFile – Tools for maintaining ASCII program files, LabVIEW datalog files, and downloading files to PMAC.

\PmacPQM – PMAC program execution is parametrically specified using P, Q, and M-Variables. For example, the number of times a move is executed, the increment of a move, or the radius of a circular move can all be specified using Ps and Qs. Specific machine inputs and outputs, and internal registers are accessible using M-Variables. Mapping of these quantities to LabVIEW controls is facilitated by the ICVs in this collection. In addition, the ability to log this information to a LabVIEW datalog file and re-execute the motion at a later time is provided.

\PmacPLC – PLC programs and their execution status can be edited and controlled using the VIs in this collection.

\PmacProgram – This collection provides tools at a variety of levels to edit, download, debug, monitor, and encapsulate motion programs. Encapsulated programs:

- P, Q, and M variant Load themselves when executed
- Know their coordinate system and program number
- Can be executed by the click of a button
- Indicate the state of their execution
- Can be modified, monitored, and debugged from a powerful front panel
- Accept data types from the \PmacPQM tools

\PmacSubVI – This directory contains the actual encapsulated program and PLC wrapper template VIs.

Data Gathering and Graphical Tools

One of PMAC's most intriguing capabilities is its ability to synchronously gather a variety of motion data during the execution of a program or move. An example is the gathering of actual and desired position in response to step inputs. This data can be used to analyze the performance of a specific move or machine configuration using LabVIEW's powerful analysis suite.

\PmacGather – This collection of VI's allow the user to select motion variables to control the collection and conversion of data into standard LabVIEW analysis formats. Collected data can be output to files in tab delimited format for export to programs such as Matlab or Excel.

\PmacAddress – A collection of tools for specifying addresses, scale factors, and descriptions for gathering. Variables not already accessible from PMAC Panel Pro can easily be added to the tables.

\PmacPlot – A few generally useful plotting VIs for setting plot colors and legends. An XY Chart buffer is available to make an XY plot into an XY strip chart.

Code Interface Nodes and Dual-Ported RAM

PMAC Dual-Ported RAM (DPR) provides a high-speed binary data transfer mechanism that speeds access to certain types of motion data. To facilitate this capability, PMAC Panel Pro utilizes LabVIEW Code Interface Nodes (CINs). This collection of VIs demonstrates how to use this capability.

\PmacCIN – A detailed description of the process required for configuring the environment to use CINs with PMAC Panel Pro.

\PmacDPR – A large collection of VIs for configuring and operating the many modes of DPR supported by PMAC. There are numerous examples demonstrating the use of DPR and how to modify the supplied collection to suit specific purposes.

\PmacInc – A directory of include files required to use CINs or modify PmacDPR. This directory will be empty if the PComm 32 Pro package was not purchased.

Sample Applications

This collection of VIs demonstrates a general PMAC Panel Pro application and a set of tutorials to illustrate the correct use of PMAC Panel Pro capabilities.

\PmacTestPRO – An all encompassing demonstration of program encapsulation and monitoring with four different motion programs, their PQM configuration, and real-time strip chart monitoring.

\PmacTutorial – A sequence of exercises covered in the Application Basics section of this manual. These exercises show the basic architecture and proper use of PMAC Panel Pro in individual applications. All first time users of PMAC Panel Pro should read this section and examine the tutorial VIs.

\PmacDAQ – This collection of VIs utilize standard LabVIEW analog input DAQ example VIs and a PMAC Panel Pro motion VI to demonstrate a few of the techniques thatcan be used to integrate PMAC and NI-DAQ boards to develop precision motion based data acquisition applications.

Miscellaneous Utilities

This collection provides many VIs used to implement PMAC Panel Pro without regard to a specific category.

\PmacSetup – True maintenance of a deployable PMAC application requires Pewin 32 Pro. This collection of VIs is the start of a set of VIs to download and maintain the configuration of PMAC's numerous P, Q, M, and I-Variables. For the purpose of speed, the VIs are implemented using CINs.

\PmacUtility – This collection provides controls and VIs to modify file paths, handle radio buttons, etc.

Documentation

PMAC Panel Pro has an extensive suite of documentation and help that can answer most questions. This manual along with the various PMAC User and Reference manuals contains thousands of pages of information on every aspect of PMAC and PMAC Panel Pro. There are numerous help files available and more being written all the time. Check the Delta Tau web site for documents and help files for a specific system.

\PmacDocument – Contains electronic copies of this document, miscellaneous help items, and a few useful bitmaps.

APPLICATION BASICS

Basics

This chapter contains several systematic exercises that run through the hierarchy of PMAC Panel Pro to introduce the various concepts required to develop custom applications. For detailed explanations of the individual VIs used in each tutorial, consult the VI Reference. First time users of PMAC Panel Pro must read this section.

Although most PMAC Panel Pro VIs can be used as is, the developer is encouraged to use them as templates that can be extended and customized to meet the requirements of an individual application.

LabVIEW Techniques for PMAC Panel Pro

The following are general LabVIEW programming techniques not related to a specific PMAC Panel Pro VI and therefore may not appear elsewhere in this manual. It assumes that the user has taken a basic course in the use of LabVIEW or has equivalent experience.

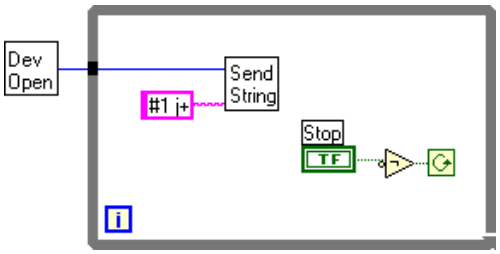
PMAC Panel Pro's architecture was designed to hierarchically encapsulate common operations into VIs that it can be used to develop custom applications without doing a lot of basic communication parsing and wiring. Depending on the application's requirements and the user's experience with LabVIEW, there will be custom design patterns. The tutorial exercises that follow and the examples contained in the release reflect different ways to architect the applications to maximize the utility of PMAC Panel Pro. These techniques are used throughout PMAC Panel Pro and can be applied in individual applications. For an excellent reference on LabVIEW techniques and application design issues, see LabVIEW Graphical Programming by Gary Johnson (McGraw Hill ISBN 0-07-032915-X).

Dataflow and Sequencing

In LabVIEW the order of VI, execution is not guaranteed. Some PMAC operations require sequenced command execution. For example, a command to start a motion program should be sent before waiting for it to complete. In other situations, M-Variables must be defined before they can be used. Some PMAC Panel Pro VIs anticipate the need for sequencing and provide an output to enable this without using a LabVIEW sequence structure. In some instances, a sequence structure must be used.

Dataflow and Recurring Execution

LabVIEW loves to use while loops to execute VIs. To continually send a **JOG+** command to PMAC, use the following example. It will send the command repeatedly. This wastes PMAC's time and slows the PMAC Panel Pro application. To prevent this, develop program logic so that the commands are not repeatedly sent to PMAC. Most of PMAC Panel Pro's architecture is designed to simplify this by encapsulating this logic at the lowest levels possible.



Giving up Control

Make sure that the PMAC VIs using concurrent execution loops use a wait timer to give the user interface and other VIs a fair shot at executing. Otherwise, the user interface may lockup while the PMAC Panel Pro program is waiting for a motion program to finish. Worse yet, other VIs that use double buffering to acquire data can overflow with nasty results.

Execution Speed

LabVIEW is fast at doing certain types of things and slower at others. Writing complex VIs such as the terminal is tough on LabVIEW execution and the developer. Do not expect the response time of visual C++ or Pewin 32 Pro.

Realize that applications that repeatedly poll PMAC can slow the application. This is true especially of systems that use serial communication at low baud rates. On the other extreme, there are busy applications that use the DPR capabilities of PMAC Panel Pro for high-speed data transfers.

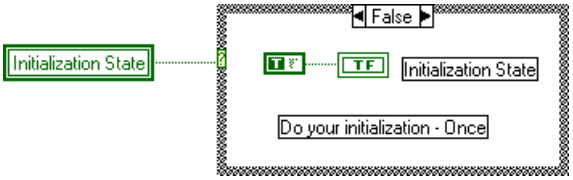
VI Reentrancy

In general, most VIs are reentrant. In LabVIEW terminology, this means that a VI can be used simultaneously in multiple PMAC Panel Pro and application VIs with its own separate copy of data. This has some benefits and some drawbacks. It allows independent simultaneous execution of the reentrant VI. The drawback is that it prevents each use of the VI from having a user accessible copy of the panel. In the case of low level PMAC Panel Pro VIs this is not really an issue because the VIs have no user interface value. It becomes more of an issue with the ICV VIs.

Issues can also arise when sending commands to PMAC. If multiple VIs are busy sending commands, it is not only possible, but also probable that expected responses will not line up with the commands. This is covered later in this section.

Persistent VI State

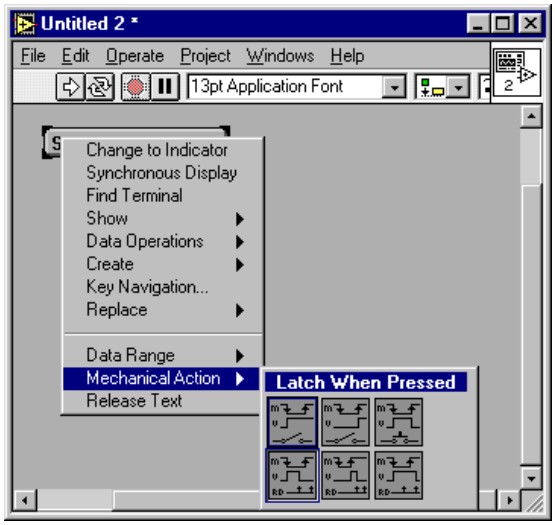
LabVIEW VIs maintain their state from execution to execution within a loop as long as they are loaded in memory. Sometimes this is desirable, and in many instances is used by PMAC Panel Pro Vis, particularly those that are used to build configuration tables or attempt to minimize repetitive queries for data that will not change often. The following diagram demonstrates the use of this technique:



In this example, the default value for Initialization State is false. It is set to this value every time the VI is loaded into memory. When executed the first time, Initialization State is set to true and will remain that way until it is reset by an operation in this VI or the calling VI is closed thereby unloading this from memory. Be aware that LabVIEW 5.0 has a bug handling this. National Instruments plans to release a fix for this problem with 5.0.1. Until then, PMAC Panel Pro 1.0 has added a few wires and local variables to fix this.

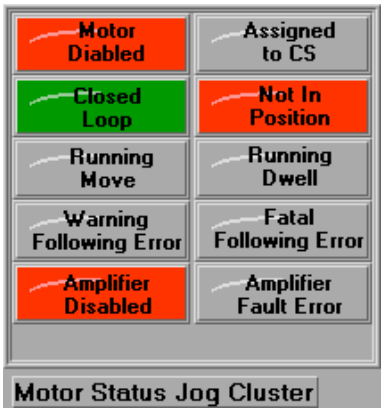
Mechanical Action

Most PMAC Panel Pro VIs requiring Boolean inputs or buttons have their mechanical action set to Latch When Pressed. Configure the buttons as shown in the following window. Using this configuration, the button is read once when clicked and reset to false. This is useful in preventing commands from being repeatedly sent to PMAC.



PMAC Panel Pro Indicator and Control Clusters

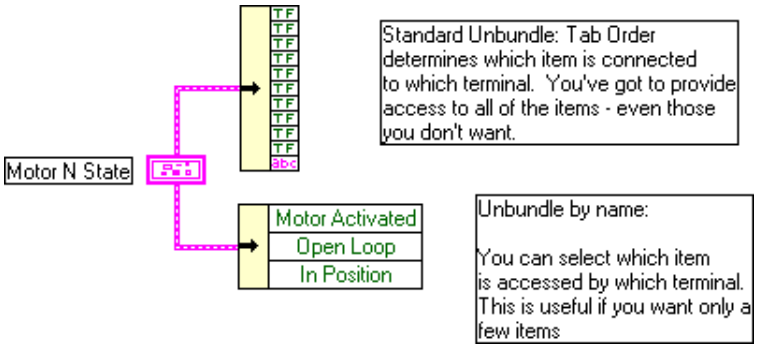
PMAC Panel Pro makes extensive use of predefined indicator and control clusters to make the development task easier. These clusters are easy to drop into an application. It is necessary to understand what clusters are, how to edit them, and how to access the individual controls and indicators they contain. An example of a Motor Status Jog Cluster PmacMotorStatJog is shown here:



A lot of work went into developing indicator clusters with proper names and item descriptions. When **Help»Show Help** is enabled, extensive descriptions of PMAC Panel Pro cluster items can help the user understand what is displayed and what operations are performed by controls and indicators.

Cluster Item Access

The individual items of each PMAC Panel Pro cluster are named and given a Tab Order. Within PMAC Panel Pro, generally they are unbundled without the name so that the diagrams are a little easier to fit on a single page. The diagram below demonstrates the two different techniques for accessing the individual items of a cluster.

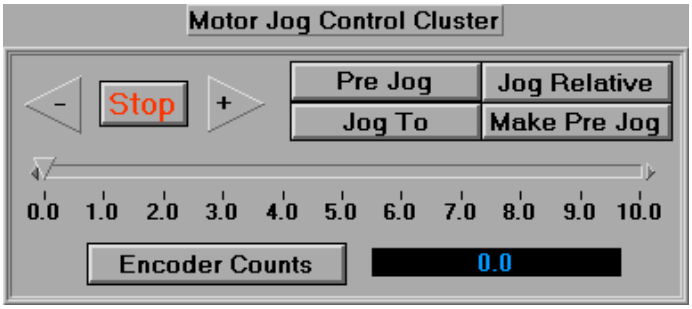

Motor N State
Motor Activated
Open Loop
In Position
Standard Unbundle: Tab Order determines which item is connected to which terminal. You've got to provide access to all of the items - even those you don't want.
Unbundle by name:
You can select which item is accessed by which terminal. This is useful if you want only a few items

Using the mouse on function VI terminals, cluster constants can be created easily if required. These can then be filled with the appropriate data.

Clusters Contain Controls or Indicators but not both

Clusters are wonderful for grouping commonly used items together to make it easier. A major limitation should be understood. In general, a cluster should not contain both indicators and controls. This does not work well with LabVIEW's data flow execution model - what happens if it sets an indicator item before it reads it as a control value? This is true especially of PMAC Panel Pro clusters that use Booleans configured with latched mechanical action. LabVIEW will not let this possible race condition go and will generate an error.

PMAC Panel Pro has made some concessions for this. The PmacMotorJogControl control cluster shown below can be used to jog a motor. A separate position cluster has been created.

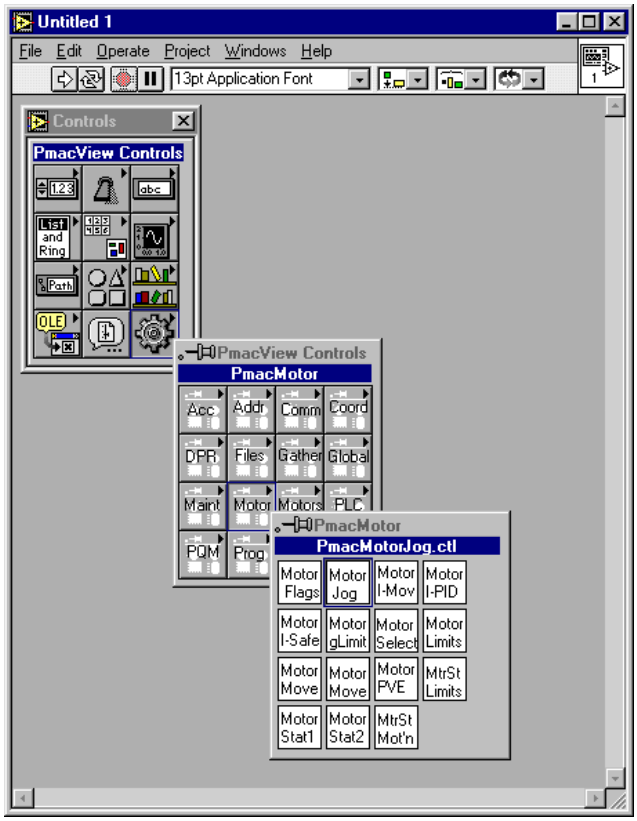


Cluster Type Definitions

LabVIEW allows the defining of cluster controls and indicators as strict types that are linked to the root control definition. PMAC Panel Pro has chosen not to use this capability. Clusters can be used and modified independent of the base definition. This means that if changing cluster definitions, the changes are not propagated to the VIs using it. All instances of the control must be replaced by hand. Or choose to define the *.ctl controls as strict types before using them. Then every new instance of the control in the application will be linked to the raw control.

Accessing PMAC Panel Pro VIs

To use PMAC Panel Pro Vis, select the PMAC Panel Pro control sub-palette attached to the PMAC Panel Pro.lib directory. Depending on the view installed, the following screen may appear.

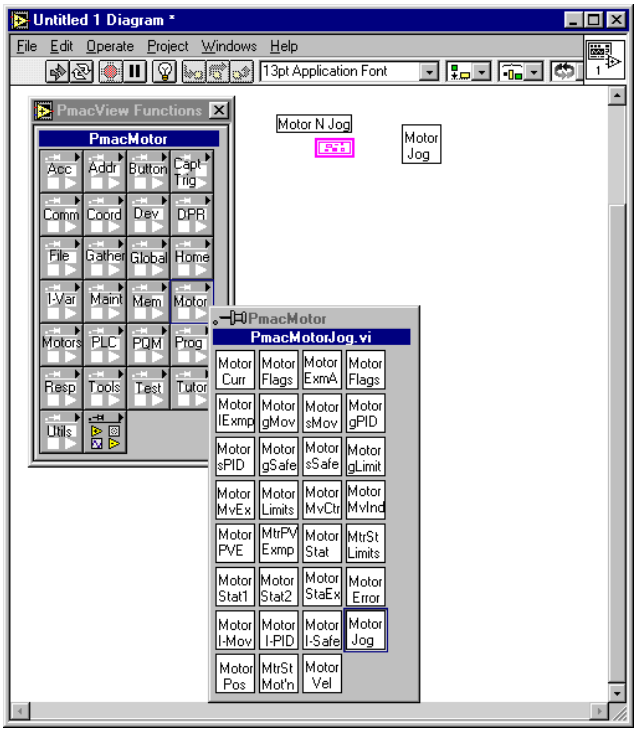


PMAC Panel Pro sub-palettes use words rather than graphical icons to define the functionality just as the naming in the directory structure does. To place a PMAC Panel Pro Control on the panel, the icon is selected from the palette and placed on the VIs panel.

Clusters with an Associated Function VI

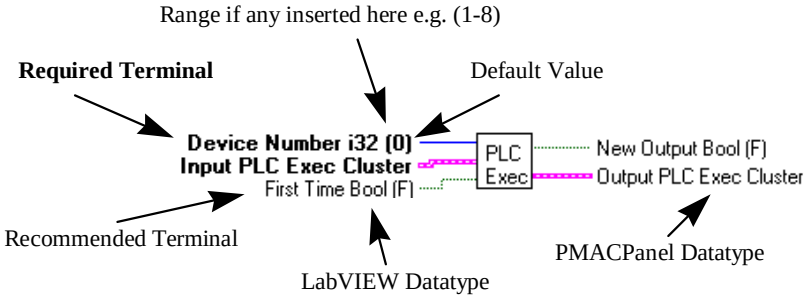
PMAC Panel Pro controls exist on VI Panels. To get the data for indicators or generate commands from controls, they need a PMAC Panel Pro function VI. To make it easier to link the two together, the name of the associated function VI is the same as that of the control. Details on this are outlined in the next section.

The screen shown below shows the terminal for PmacMotorJogControl.ctl and the function VI PmacMotorJogControl.vi. The similar names indicate that they are paired together and the panel cluster is wired somewhere on the function VI icon.



PMAC Panel Pro VI Terminal Conventions

PMAC Panel Pro has carefully chosen terminal names that are used consistently throughout the library. Extensive terminal descriptions and their behavior are available with LabVIEW's **Help»Show Help** facility and in the VI Reference chapter of the manual. To help utilize the on-line help associated with PMAC Panel Pro Vis, there are terminal naming conventions to identify the name, type, default value, and range of inputs and outputs. Most input terminals have a default value. Those that have a range coerce inputs to the range. Most inputs are required. A few are recommended. Their default action is fully defined. Typically, cluster inputs and outputs are named for the wire type expected. The following VI icon demonstrates these standards.



PMAC Panel Pro Tutorials

If the basic ideas covered in the previous section are understood, start developing PMAC Panel Pro applications. The remainder of this section contains exercises that start with opening communication with PMAC and work up to the development of some very sophisticated capabilities. The contents of these tutorials are necessary for all PMAC Panel Pro developers. The on-line Windows Help version of this document is a great way to display the contents of the tutorials and examples from within LabVIEW.

Tutorial VIs are located in the \PmacTutor directory. In this directory, there are two base VIs that are used as a starting place for developing all the examples and PMAC Panel Pro VIs. These are

- **PmacTutorApp** – Opens PMAC, has an execution While loop, and a Stop button
- **PmacTutorSub** – Has a predefined Device Number terminal

The tutorial VIs are named PmacTutor1, PmacTutor2, etc. These should be opened, examined, and executed while working through the exercises. Save new copies of these and modify them as desired.

Depending on the exercise goals, various panels, diagrams, and sub-VI descriptions are used to illustrate PMAC Panel Pro concepts. Numerous descriptions are provided on the panels and in the diagrams.

PMAC Panel Pro Communication Tutorial

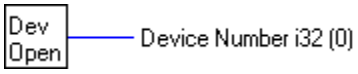
The following exercises introduce those VIs used to send commands to PMAC and access its data. This includes special collections of VIs for:

- PComm 32 Pro access
- Sending commands
- Querying PMAC and processing responses
- Accessing PMAC memory-mapped data and variables
- Querying and Setting I-Variables

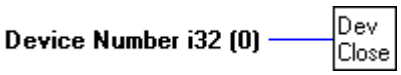
PmacTutor1 – Accessing PComm 32 Pro

PmacTutor1 covers the basic requirements for accessing PComm 32 Pro. All PMAC Panel Pro applications must open access to PComm 32 Pro using PmacDevOpen. This exercise demonstrates three steps for all PMAC Panel Pro applications. Open the device, do something, close the device. These VIs are:

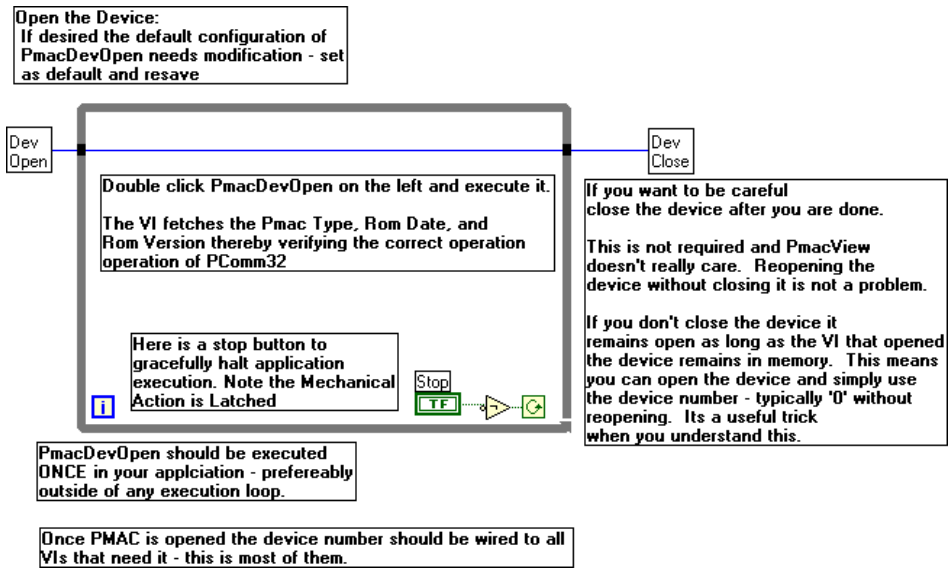
- PmacDevOpen - Open communication to PMAC using the PComm 32 Pro device driver. Check type, ROM date, and ROM Version. Provide Device Number for other VIs. Select the mode of communication using the Communication Mode drop down menu. To make the selection permanent, make the selection the default, use the right mouse button and select the **Data Operations»Make Current Value Default** option. This must be done in conjunction with the options available on the PMAC control panel.



- PmacDevClose - Close the PComm 32 Pro device driver. PMAC will continue running as programmed as long as power is applied.



All basic PMAC Panel Pro application diagrams will open PMAC and pass the device number into the application's main loop where the primary logic is executed. Structuring the main loop this way establishes a dependency between the opening of the device and the execution of the rest of the application.


Open the Device:
If desired the default configuration of PmacDevOpen needs modification - set as default and resave
Dev Open
Dev Close
Double click PmacDevOpen on the left and execute it.
The VI fetches the Pmac Type, Rom Date, and Rom Version thereby verifying the correct operation operation of PComm32
If you want to be careful close the device after you are done.
This is not required and PmacView doesn't really care. Reopening the device without closing it is not a problem.
If you don't close the device it remains open as long as the VI that opened the device remains in memory. This means you can open the device and simply use the device number - typically '0' without reopening. Its a useful trick when you understand this.
Here is a stop button to gracefully halt application execution. Note the Mechanical Action is Latched
Stop
TF
PmacDevOpen should be executed ONCE in your applciation - prefereably outside of any execution loop.
Once PMAC is opened the device number should be wired to all VIs that need it - this is most of them.

Several techniques to ease the development process are noted in the diagram descriptions. These should not be relied upon in the final application.

PMAC Panel Pro was designed to be easy to use. To avoid having to provide PmacDevOpen with a device number, communication mode, etc. every time it is used, the VI was configured with defaults for the device number and communication mode in the PMACPanel Basics section of this manual. If using more than one PMAC or to access the device using more than one mode, use one of two equally simple methods:

1. Make copies of PmacDevOpen and rename the copies (e.g. PmacDevOpenSerial or PmacDev0Open and PmacDev1Open). Set the defaults for the device number and communication mode as desired. Use these exactly as PmacDevOpen would be used.
2. Make a copy of PmacDevOpen and add input terminals for the device number and/or communication mode. When using this VI provide the inputs and use the outputs exactly as PmacDevOpen would be used.

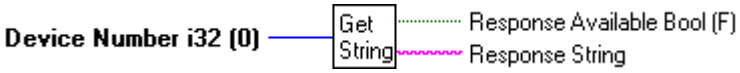
Multi-threading and PmacDevOpen

PMAC Panel Pro and PComm 32 Pro readily make use of LabVIEW's multi-threaded programming model. There can be multiple PMAC Panel Pro application VIs open to PMAC and execute simultaneously without problems.

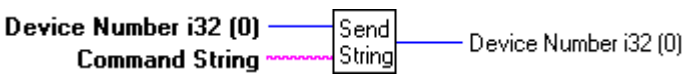
PmacTutor2 – Sending Commands and Getting Responses

The most basic interaction with PMAC is done using one of three PmacComm Vis:

- PmacCommGetStr - Check if PMAC has data available. When Response Available is true, Response String contains all available data. When Response Available is false, Response String is the empty string. Responses are parsed for PMAC ERR codes and flagged with a modal dialog.



- PmacCommSendStr - Send Command String to PMAC. If command string is the empty string, nothing is sent. The output device number is a copy of input device number to allow sequencing of commands to PMAC.



- PmacCommRespStr - Send Command String to PMAC and wait for a response. If Command String is the empty string, nothing is sent. Response Available is true when Response String contains response data. When Response Available is false, Response String is the empty string. Responses are parsed for PMAC ERR codes and flagged with a modal dialog.

All the PMAC on-line commands described in the PMAC User Manual and PMAC Software Reference Manual are valid commands. See the appropriate manual for detailed command usage and syntax. PMAC will accept multiple commands in a command string. Those commands that generate a response will put the data into PMAC's output buffer whether or not it is retrieved. If a command generates a response, use PmacCommRespStr.

Generally, PMAC responses are single lines. The exceptions to this are commands such as:

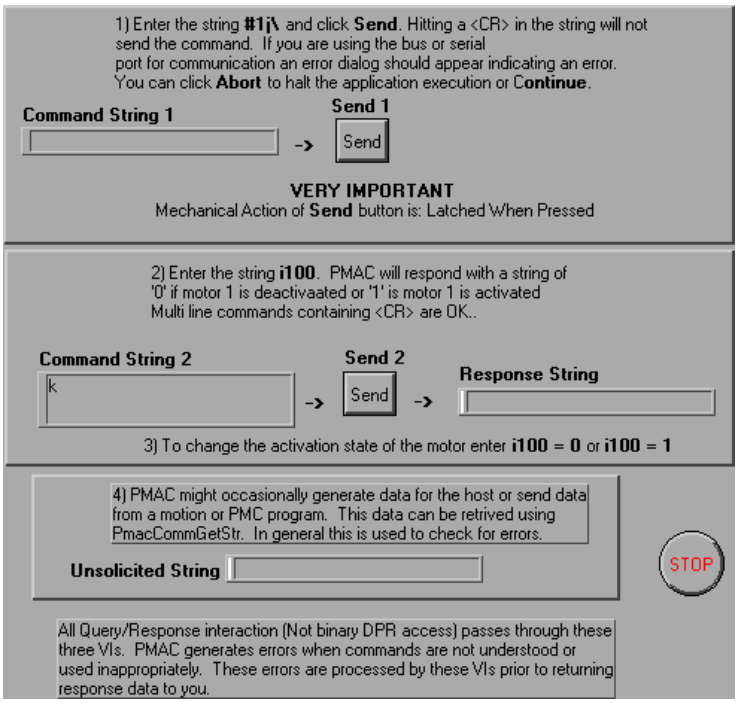
```
LIST PROG n
LIST GATHER
```

Note:

LabVIEW string controls do not treat a **<CR>** as anything other than a **<CR>**. It is possible to tie this keystroke to a control on the panel. Check the LabVIEW documentation on Key Navigation Option for Controls.

These commands will generate long multi-line responses. LabVIEW strings can handle these. Depending on the size of the string indicator or control on the panel, these may not wrap correctly. This is a LabVIEW issue. If PmacTerminal is executed and a long gather buffer is listed, this will appear. There is also an issue when entering strings using a control. The natural temptation is to expect that hitting **<CR>** will cause the string to be sent. LabVIEW does not work this way. Panel values are input using the **<Enter>** key.

The panel and diagram for this exercise demonstrate the use of these three communication VIs. Execute the operations specified in step 1 through 4 on the panel.


1) Enter the string #1j\ and click Send. Hitting a <CR> in the string will not send the command. If you are using the bus or serial port for communication an error dialog should appear indicating an error. You can click Abort to halt the application execution or Continue.
Command String 1
Send 1
Send
VERY IMPORTANT
Mechanical Action of Send button is: Latched When Pressed
2) Enter the string i100. PMAC will respond with a string of '0' if motor 1 is deactivaated or '1' is motor 1 is activated Multi line commands containing <CR> are OK..
Command String 2
Send 2
Send
Response String
3) To change the activation state of the motor enter i100 = 0 or i100 = 1
4) PMAC might occasionally generate data for the host or send data from a motion or PMC program. This data can be retrived using PmacCommGetStr. In general this is used to check for errors.
Unsolicited String
STOP
All Query/Response interaction (Not binary DPR access) passes through these three VIs. PMAC generates errors when commands are not understood or used inappropriately. These errors are processed by these VIs prior to returning response data to you.

When step 1 of this exercise is executed, the following error dialog will appear indicating a problem with the command. If the dialog does not appear, see the PMAC Communication I-Variables section in this manual to modify PMAC's communication configuration as specified.

Caution:

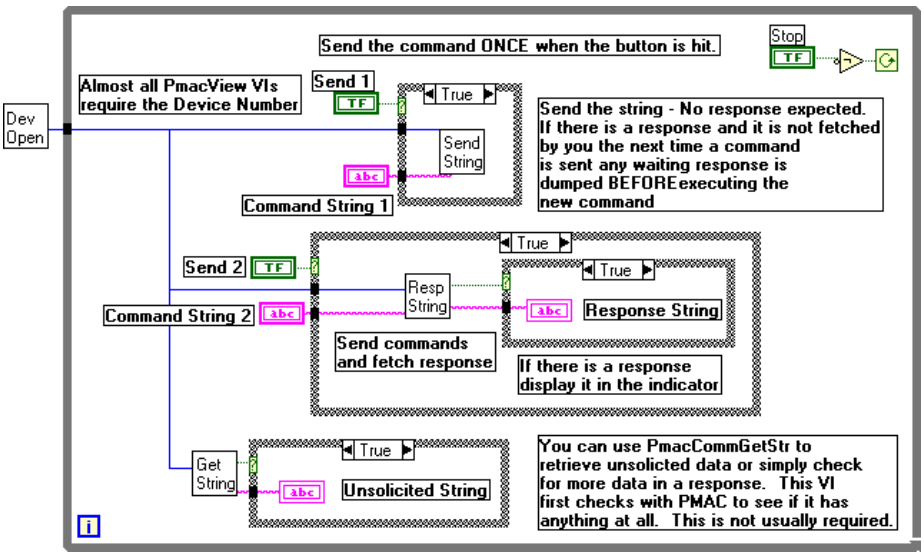
Until the proper hardware safeties have been installed, configured, and tested extreme caution must be exercised when moving motors to prevent damage and possible injury. Do not send a **Jog** command unless certain the actions will not damage someone or the system.

The error dialog appears because the command string #1j\ is unrecognizable to PMAC. The correct syntax for the command is #1j/. When error dialogs appear, choice whether to abort the application or continue. Commands that generate errors are not executed by PMAC and cause no harm. However, if the application logic continually attempts to send a bad command to PMAC, there is no choice but to abort the application. Otherwise, this dialog may be received repeatedly. The chances of halting the program using the standard LabVIEW STOP button before the application attempts to send the offending command again is unlikely. A complete listing of PMAC error codes can be found in the PMAC User Manual and PMAC Software Reference Manual.

The diagram for this tutorial demonstrates two basic things to consider when designing the application.

If commands are sent to PMAC in response to an interface button click, the mechanical action of the button should be latched and the PMAC Panel Pro VI that sends the command should be in a Case structure. Otherwise, the command will be sent every iteration of the loop.

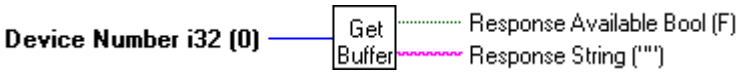
PmacCommRespStr and PmacCommGetStr indicate whether they received a response using Response Available. Response String will be the empty string if nothing was received. Generally, Response Available should be tested before using Response String.


Send the command ONCE when the button is hit.
Stop
Almost all PmacView VIs require the Device Number
Send 1
True
Dev Open
Send String
Command String 1
Send the string - No response expected. If there is a response and it is not fetched by you the next time a command is sent any waiting response is dumped BEFORE executing the new command
Send 2
True
True
Resp String
Command String 2
Response String
Send commands and fetch response
If there is a response display it in the indicator
Get String
True
Unsolicited String
You can use PmacCommGetStr to retrieve unsolicted data or simply check for more data in a response. This VI first checks with PMAC to see if it has anything at all. This is not usually required.

Using these three basic VIs an entire, albeit complex, PMAC Panel Pro application can be generated. The purpose of most of PMAC Panel Pro is to prevent doing this.

PMAC and PComm 32 Pro limit basic responses to 256 characters. PmacCommRespStr and PmacCommGetStr handle this internally using the VI PmacCommGetBuffer to retrieve longer responses. Generally, the applications will not make use of this VI.

- PmacCommGetBuffer - Check if PMAC has data available. When Response Available is true, Response String contains all available data. When Response Available is false, Response String is the empty string. Responses are not parsed for PMAC ERR codes.



Exercise PmacTutor3 shows how PMAC Panel Pro simplifies the sending of commands in response to panel buttons and Boolean conditions.

PmacTutor2a – Communication Logging

This is a rather advanced topic, but one that was included here for completeness. PMAC Panel Pro has the ability to log all communication between the application and PMAC. This might be used to monitor what users are doing or to log interesting sessions for later play back. PMAC Panel Pro maintains all communication using the following VIs. Future versions of PMAC Panel Pro may use this capability to implement a Graphical Motion Language.

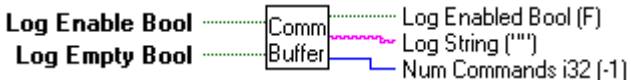
- PmacCommGlobal - This VI is a global copy of the PmacCommGlobal cluster used by several PmacComm VIs for error reporting and logging purposes.

Pmac Communication Cluster (This cluster maintains a log of communication between PMAC Panel Pro and PMAC.)

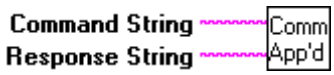
- **Command String** (Last on-line command sent to PMAC)
- **Response String** (Last response received from PMAC)
- **Communication Log String** (A multi-line buffer of commands sent to PMAC and received from PMAC.)
- **Num Commands i32** (The number of commands sent to PMAC and logged in the Communication Buffer.)

Buffer Log Bool (F) (When true, all communication is appended and logged to the Communication Log.)

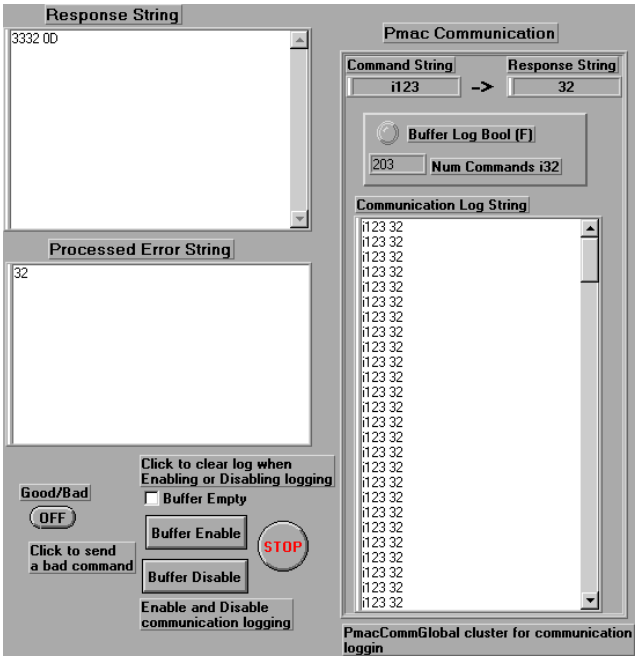
- PmacCommBuffer - When Log Enable is true, communication logging is enabled. Log Enabled Bool, Log String, and Num Commands reflect the state of the log buffer when logging is enabled. Log String is the empty string, Num Commands = -1, and Log Enabled is false when logging is disabled. When Log Empty is true, the log buffer is emptied.



- PmacCommAppend - Copy Command String and Response String to the Last Communication items in PmacCommGlobal. If Logging Enabled is true they are also appended to the Communication Log.

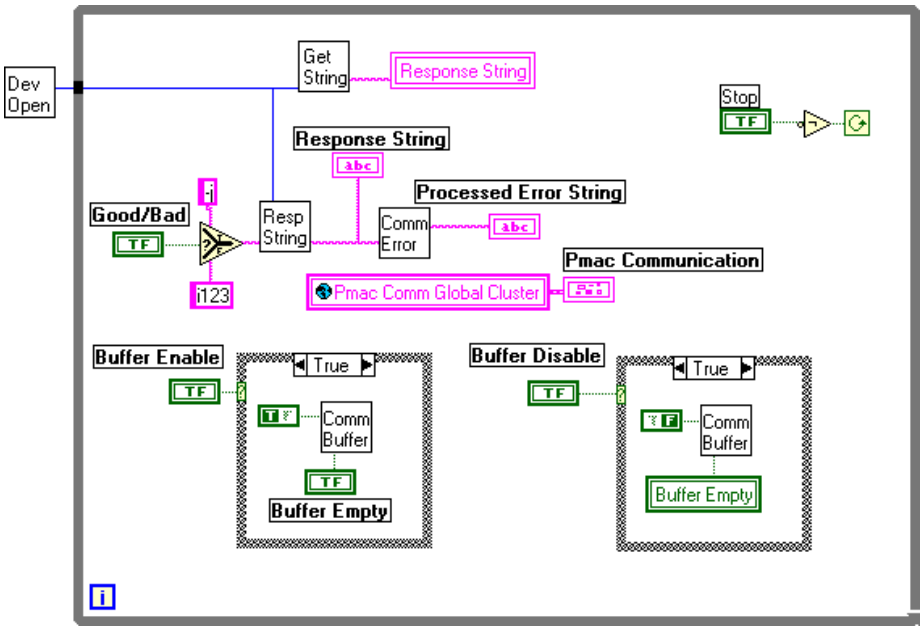


The panel for this exercise demonstrates how communication is logged. The VI queries PMAC for the value of i123. When Good/Bad is clicked, the incorrect command **`-j`** is sent resulting in an error. The log of this bad command appears in the Command String and Response String items in the PMAComunication Cluster on the right. The error dialog pop-up will also appear allowing to continue or abort the application. Click **Continue**.



If the **Buffer Enable** button is clicked, all communication is appended to the Communication Log Buffer in the cluster. To stop logging, click **Buffer Disable**. This only stops the logging of communications. If the **Buffer Empty** box and **Buffer Enable** is clicked, the buffer is cleared before logging is enabled.

The diagram for this exercise is shown here:

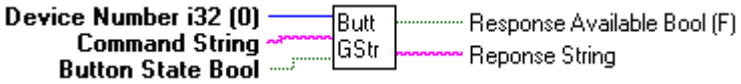


To use logging in the application, develop logic to save the Log Buffer to a file. Realize that the size of this buffer could grow very large if the application uses status-monitoring ICVs and the contents of the log buffer is not saved and emptied at reasonable intervals.

PmacTutor3 – Sending Commands Using Buttons

PMAC Panel Pro contains the PmacButton collection of VIs that send a command string to PMAC when an input button state is true. The applications panels will make constant use of these capabilities.

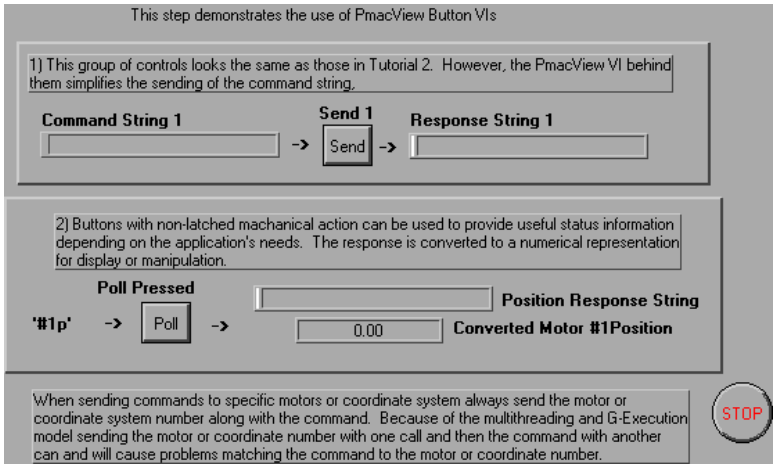
- PmacButtGetStr – Send Command String to PMAC and wait for a response when Button State is true. When Response Available is true, Response String contains the response. If Response Available is false, Response String defaults to the empty string.



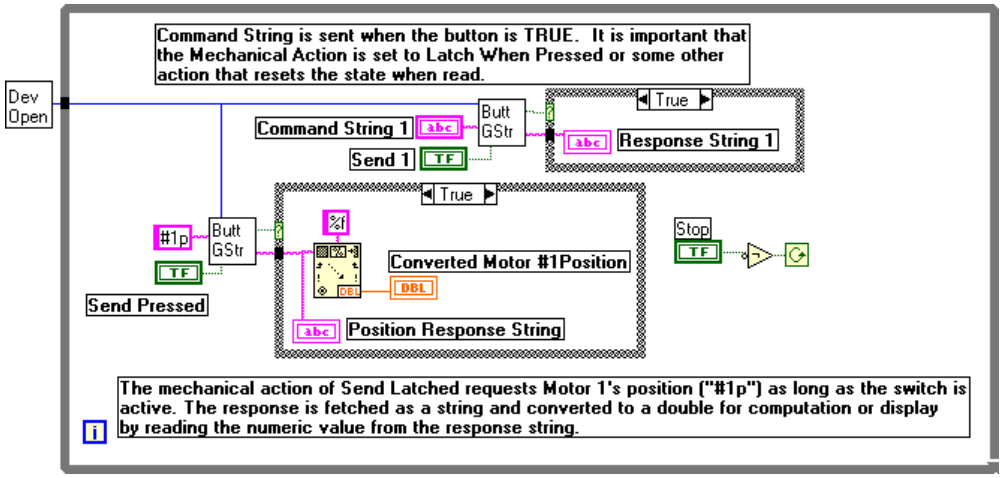
- PmacButtSendStr – Send Command String to PMAC when Button State is true. Response Available is true when PMAC has processed the command.

Use PmacButtGetStr for commands that expect responses and PmacButtSendStr for commands that do not expect responses. PmacButtSendStr does not return a response so the input Button State is passed through to facilitate execution sequence dependencies.

The panel shown here demonstrates the use of these VIs, the conversion of PMAC responses into numeric data for use in LabVIEW, and the use of non-latched mechanical action to enable polled status for real-time.


This step demonstrates the use of PmacView Button VIs
1) This group of controls looks the same as those in Tutorial 2. However, the PmacView VI behind them simplifies the sending of the command string.
Command String 1
Send 1
Send
Response String 1
2) Buttons with non-latched machanical action can be used to provide useful status information depending on the application's needs. The response is converted to a numerical representation for display or manipulation.
Poll Pressed
'#1p'
Poll
Position Response String
0.00
Converted Motor #1Position
When sending commands to specific motors or coordinate system always send the motor or coordinate system number along with the command. Because of the multithreading and G-Execution model sending the motor or coordinate number with one call and then the command with another can and will cause problems matching the command to the motor or coordinate number.
STOP

Note that in the diagram for this VI, the sending of the command to PMAC is simplified by the use of the PmacButtGetStr. The button is directly supplied to the VI rather than wrapping the PmacCommRespStr VI in its own case structure. Sending commands in response to buttons is so common that this added capability makes application development significantly simpler.


Command String is sent when the button is TRUE. It is important that the Mechanical Action is set to Latch When Pressed or some other action that resets the state when read.
Dev Open
Command String 1
Send 1
Butt GStr
True
Response String 1
#1p
Send Pressed
%f
Converted Motor #1Position
Position Response String
Stop
The mechanical action of Send Latched requests Motor 1's position ("#1p") as long as the switch is active. The response is fetched as a string and converted to a double for computation or display by reading the numeric value from the response string.

The second step in the diagram sends the command **#1p** to PMAC requesting the position for motor #1 as long as the button is true. The response string is displayed both as a string and converted into a numerical value for display. When sending any motor or coordinate system specific command to PMAC, include the motor or coordinate system number with the command to prevent LabVIEW's execution order from sending addresses and commands in whatever order it desires.

PmacTutor4 – Button and Response VIs

The PmacResponse VIs send commands to PMAC and convert the ASCII response string into LabVIEW numeric data types. String conversion VIs are not scattered all over the application. There are more members in the PmacButton collection introduced in PmacTutor3 that use the PmacResponse VIs to convert ASCII string responses into numeric data.

PmacResponse

PmacResponse consists of six VIs, one for each type of numeric response conversion supported. If there is a data type that must be supported, add it by copying and modifying one of these.

When using these VIs, refer to the PMAC Software Reference Manual; determine the size of the response, whether it will be signed or unsigned, and whether it will be manipulating the bits of the response. PmacRespGetDbl is used to introduce the collection.

- PmacRespGetDbl – If Command String is not the empty string, send it to PMAC and wait for a response. If Response Available is true, Response contains a valid response. Otherwise, Response is 0.0.

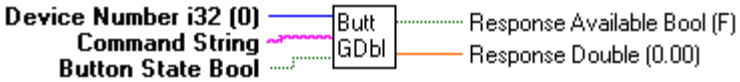
The remaining VIs operate the same and simply provide responses of the appropriate type:

- PmacRespGetBool
- PmacRespGetShort
- PmacRespGetUShort
- PmacRespGetLong
- PmacRespGetULong

PmacButton

PmacButton consists of six additional VIs beyond PmacButtGetStr and PmacButtSendStr and introduced in PmacTutor3. These additional VIs provide numerical responses. PmacButtGetDbl is used to introduce the collection.

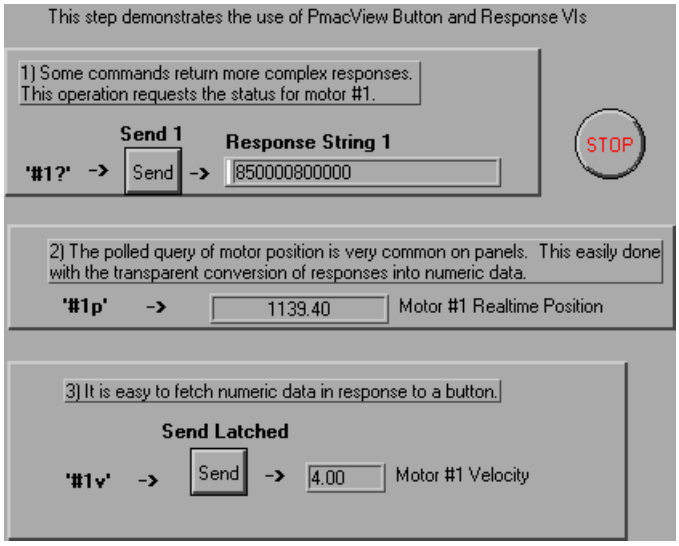
- PmacButtGetDbl – Send Command String to PMAC and wait for a response when Button State is true. When Response Available is true, Response Double contains the response. If Response Available is false, Response Double defaults to 0.0.



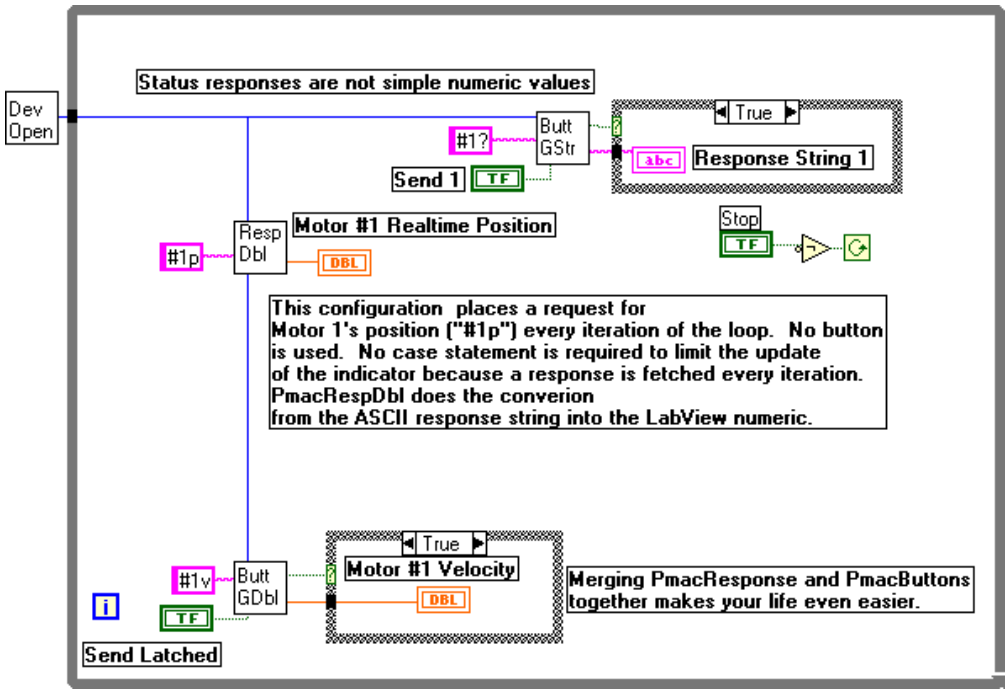
The remaining VIs operate the same and simply provide responses of the appropriate type:

- PmacButtGetBool
- PmacButtGetShort
- PmacButtGetUShort
- PmacButtGetLong
- PmacButtGetULong

Step 1 of the exercise demonstrates that some response data is a little more than a numerical value. Step 2 demonstrates how a single PmacResponse VI can be used to provide a useful piece of data for the panel. When coupled with the button concept, PMAC data can be requested and converted using a single VI.


This step demonstrates the use of PmacView Button and Response VIs
1) Some commands return more complex responses.
This operation requests the status for motor #1.
Send 1
Response String 1
'#1?' ->
Send
->
850000800000
STOP
2) The polled query of motor position is very common on panels. This easily done with the transparent conversion of responses into numeric data.
'#1p' ->
1139.40
Motor #1 Realtime Position
3) It is easy to fetch numeric data in response to a button.
Send Latched
'#1v' ->
Send
->
4.00
Motor #1 Velocity

Although the operations are more complex, the diagram is simpler.

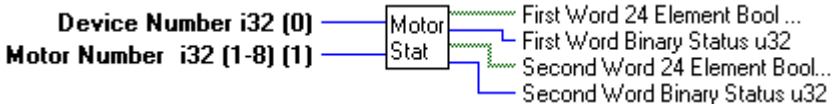

Status responses are not simple numeric values
Dev Open
#1?
Butt GStr
True
abc
Response String 1
Send 1
TF
Stop
TF
Resp Dbl
#1p
DBL
Motor #1 Realtime Position
This configuration places a request for Motor 1's position ("#1p") every iteration of the loop. No button is used. No case statement is required to limit the update of the indicator because a response is fetched every iteration. PmacRespDbl does the converion from the ASCII response string into the LabView numeric.
#1v
Butt GDbl
TF
True
Motor #1 Velocity
DBL
Merging PmacResponse and PmacButtons together makes your life even easier.
Send Latched

PmacTutor5 – Accessing PMAC Status

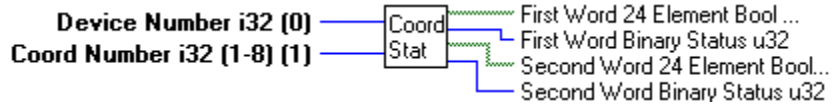
PmacTutor4 demonstrated the conversion of PMAC responses into numerical LabVIEW data types. Several responses require conversions that are more sophisticated. PMAC status is returned as twelve hexadecimal characters for a total of two 24-bit status words. Because status is critical to successful integration of PMAC with the LabVIEW application, there are three VIs that request status and convert it into two unsigned integers and two 24-element Boolean arrays. The Boolean array representations allows selection of individual status bits for custom use using LabVIEW index VIs. The unsigned integers can be used for individual bit manipulation and testing using logical operators.

These VIs are:

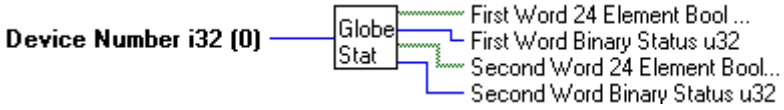
- \PmacMotor\PmacMotorStat - Query PMAC for the status of Motor Number. Report the two status words as arrays of Booleans and unsigned 32-bit integers.


Device Number i32 (0)
Motor Number i32 (1-8) (1)
Motor Stat
First Word 24 Element Bool ...
First Word Binary Status u32
Second Word 24 Element Bool...
Second Word Binary Status u32

- \PmacCoord\PmacCoordStat - Query PMAC for the status of the CS specified by Coord Number. Report the two status words as arrays of Booleans and unsigned 32-bit integers.


Device Number i32 (0)
Coord Number i32 (1-8) (1)
Coord Stat
First Word 24 Element Bool ...
First Word Binary Status u32
Second Word 24 Element Bool...
Second Word Binary Status u32

- \PmacGlobal\PmacGlobalStat - Query PMAC for PMAC's global status. Report the two status words as arrays of Booleans and unsigned 32-bit integers.


Device Number i32 (0)
Globe Stat
First Word 24 Element Bool ...
First Word Binary Status u32
Second Word 24 Element Bool...
Second Word Binary Status u32

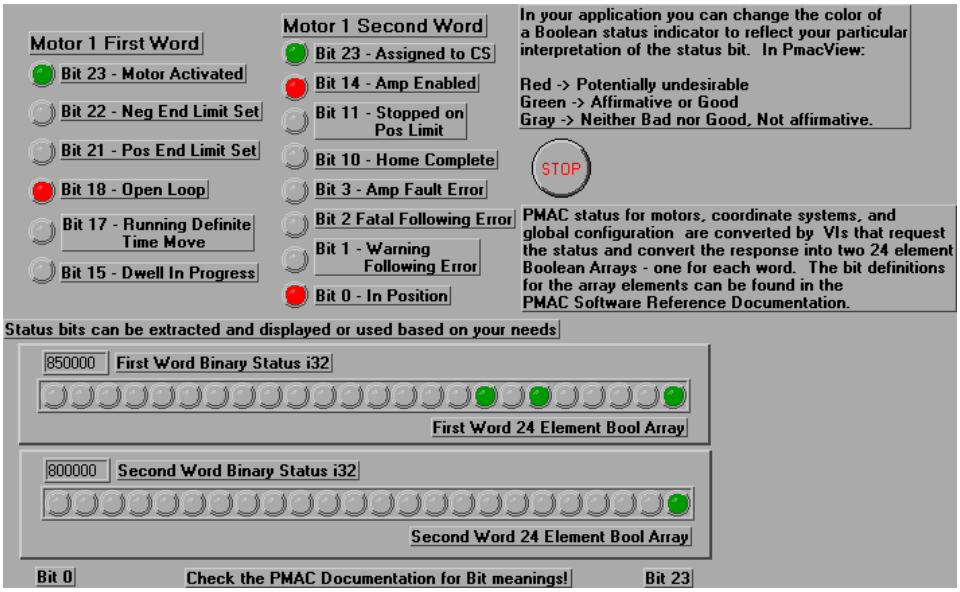
Global status does not require a motor number or coordinate system number. The Motor Number and Coord Number inputs coerce the range to protect you from mistakes. Status bit definitions can be found in the PMAC Software Reference Manual. More status processing VIs are introduced in later exercises.

Note:

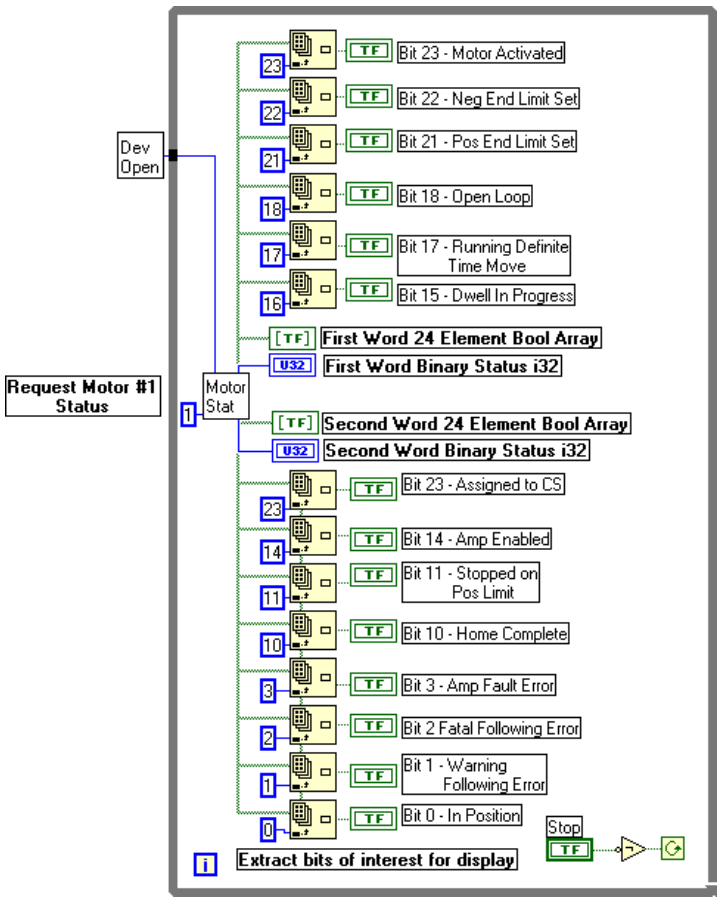
PMAC supports up to eight motors and coordinate systems. If a PMAC has fewer motors, PMAC ignores commands to them.

PMAC Panel Pro does not supply button versions of these VIs that fetch status when the button input is true. Generally, status monitoring is not a user-driven operation.

The panel for this exercise demonstrates the use of PmacMotorStat. Use of the coordinate system and global status VIs is identical. At the bottom of the panel are the raw status Boolean arrays. Because spcify Boolean text for individual array elements cannot be specified, they are left unlabeled. The contents of these arrays can be indexed using standard LabVIEW array function VIs to select specific bits for any needs. In this example, several common status bits are used to drive indicators.


Motor 1 First Word
Bit 23 - Motor Activated
Bit 22 - Neg End Limit Set
Bit 21 - Pos End Limit Set
Bit 18 - Open Loop
Bit 17 - Running Definite Time Move
Bit 15 - Dwell In Progress
Motor 1 Second Word
Bit 23 - Assigned to CS
Bit 14 - Amp Enabled
Bit 11 - Stopped on Pos Limit
Bit 10 - Home Complete
Bit 3 - Amp Fault Error
Bit 2 Fatal Following Error
Bit 1 - Warning Following Error
Bit 0 - In Position
In your application you can change the color of a Boolean status indicator to reflect your particular interpretation of the status bit. In PmacView:
Red -> Potentially undesirable
Green -> Affirmative or Good
Gray -> Neither Bad nor Good, Not affirmative.
STOP
PMAC status for motors, coordinate systems, and global configuration are converted by VIs that request the status and convert the response into two 24 element Boolean Arrays - one for each word. The bit definitions for the array elements can be found in the PMAC Software Reference Documentation.
Status bits can be extracted and displayed or used based on your needs
850000
First Word Binary Status i32
First Word 24 Element Bool Array
800000
Second Word Binary Status i32
Second Word 24 Element Bool Array
Bit 0
Check the PMAC Documentation for Bit meanings!
Bit 23

Depending on needs, the extraction of individual status bits is a tedious task. Later exercises introduce PMAC Panel Pro ICVs that extract and display the most common status bits, thereby eliminating the tedium. The diagram for the example is shown here.


Dev Open
Request Motor #1 Status
Motor Stat
Bit 23 - Motor Activated
Bit 22 - Neg End Limit Set
Bit 21 - Pos End Limit Set
Bit 18 - Open Loop
Bit 17 - Running Definite Time Move
Bit 15 - Dwell In Progress
First Word 24 Element Bool Array
First Word Binary Status i32
Second Word 24 Element Bool Array
Second Word Binary Status i32
Bit 23 - Assigned to CS
Bit 14 - Amp Enabled
Bit 11 - Stopped on Pos Limit
Bit 10 - Home Complete
Bit 3 - Amp Fault Error
Bit 2 Fatal Following Error
Bit 1 - Warning Following Error
Bit 0 - In Position
Stop
Extract bits of interest for display

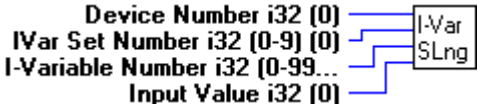
PmacTutor6 – Accessing PMAC I-Variables

On PMAC, I-Variables (Initialization, or Set-up, Variables) determine the personality of the controller for a given application. In general, this is a supervisory task. They are at fixed locations in memory and have pre-defined meanings. Most are integer values, and their range varies depending on the particular variable. There are 1024 I-Variables, from I0 to I1023, and they are organized as follows:

I-Variable Range	Functional Group
I0 -- I75	General card setup (global)
I76 -- I99	Reserved for future use
I100 -- I186	Motor #1 setup
I187 -- I199	Coordinate System 1 setup
I200 -- I286	Motor #2 setup
I287 -- I299	Coordinate System 2 setup
I3xx, I4xx, …	Motor #3, Coordinate System 3, …
I800 -- I886	Motor #8 setup
I887 -- I899	Coordinate System 8 setup
I900 -- I979	Encoder 1 - 16 setup (in groups of 5)
I980 -- I1023	Reserved for future use

To support this organization and facilitate access, PMAC Panel Pro provides a special collection of VIs to manipulate and access them. Each type of I-Variable, Boolean, Short, Long, etc. has three VIs. The VIs for accessing Long (i32) I-Variables are used to illustrate the interface.

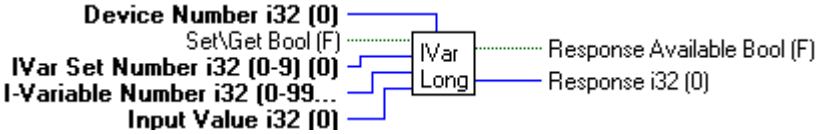
- PmacIVarSetLong - Set the Long I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



- PmacIVarGetLong - Get the Long I-Variable specified by IVar Set Number and I-Variable Number. The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.

- PmacIVarLong - If Set\Get is false or not wired, get the Long I-Variable specified by IVar Set Number and I-Variable Number. Response Available will be true to indicate Response contains the new value. If Set\Get is true, set the Long I-Variable using Input Value. Response Available will be false and Response defaults to Input Value.

The variable address is calculated as IVar Set Number * 100 + I-Variable Number. IVar Set Number = 0 addresses global I-Variables. IVar Set Numbers from 1 - 8 address motors and coordinate system I-Variables.



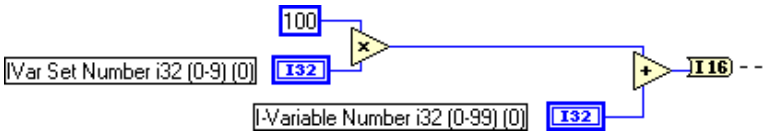
The first two I-Variable operations are obvious. The Get/Set VIs exists because when developing GUIs to configure I-Variables, they are retrieved for display and set for modification. Grouping these operations together in a single VI simplifies the diagrams. Note that the Set\Get terminal is not required. If it is not wired, the default operation for the VI is to get the I-Variable. This type of Set/Get VI architecture is common in PMAC Panel Pro.

Identical sets of VIs are provided for:

- PmacIVarDbl, PmacIVarGetDbl, PmacIVarSetDbl
- PmacIVarBool, PmacIVarGetBool, PmacIVarSetBool
- PmacIVarShort, PmacIVarGetShort, PmacIVarSetShort

There are no string I-Variables. Many of the I-Variables are bit-mapped. ICVs for collecting I-Variables into functional groups and manipulating the bit-mapped I-Variables are introduced as required later.

To access an I-Variable, the I-Variable Set Number and I-Variable Number are used to compute the number of the requested I-Variable as shown.

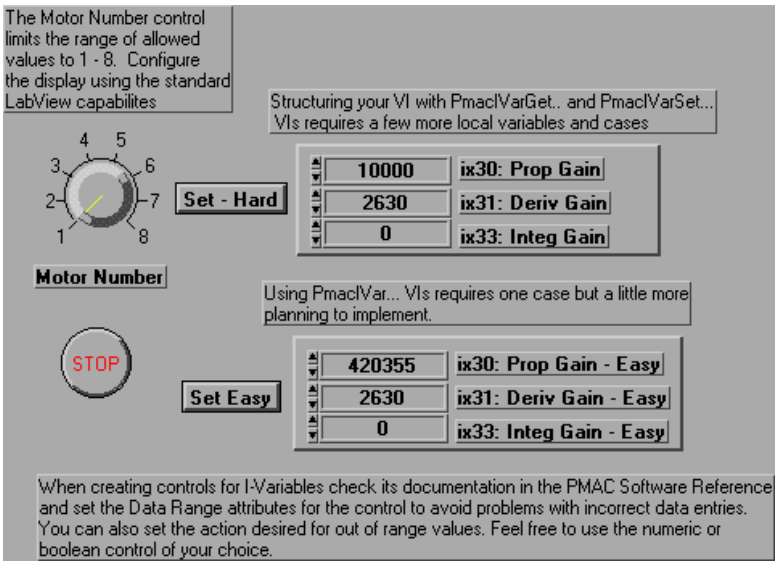


Using this approach, development of ICVs that manipulate collections of I-Variables for a particular motor or coordinate system is easy. PMAC Panel Pro does not check to see whether the I-Variable being addressed exists, its type, or its range. This information can be found in the PMAC Software Reference Manual. The I-Variable clusters introduced later perform this type of range checking where appropriate.

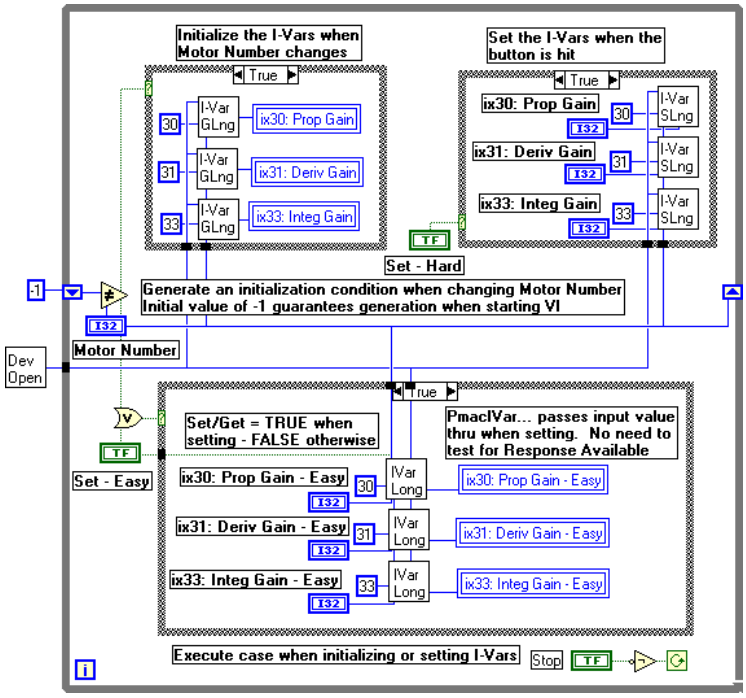
In addition to the organizational architecture, I-Variables are accessed differently by PComm 32 Pro. Use the PmacResponse VIs introduced in PmacTutor4 to access them.

The problem is that depending on the value of I9 I-Variable, queries may be returned as decimal or hexadecimal values. PMAC Panel Pro VIs in the PmacResponse collection do not support the conversion of hexadecimal responses into numerical data types. If using PmacRespGetLong to request an I-Variable like Ix25, a response of zero will be received if I9 = 2 or 3. If using PmacIVarGetLong, the proper binary representation is received along with the ability to organize access into groups.

The panel for this exercise allows the modification a few Motor PID loop I-Variables. When creating the application's VI panels, limit the range of controls to prevent potentially damaging data from being entered.


The Motor Number control limits the range of allowed values to 1 - 8. Configure the display using the standard LabView capabilites
Structuring your VI with PmacIVarGet.. and PmacIVarSet... VIs requires a few more local variables and cases
Set - Hard
10000
ix30: Prop Gain
2630
ix31: Deriv Gain
0
ix33: Integ Gain
Motor Number
Using PmacIVar... VIs requires one case but a little more planning to implement.
STOP
Set Easy
420355
ix30: Prop Gain - Easy
2630
ix31: Deriv Gain - Easy
0
ix33: Integ Gain - Easy
When creating controls for I-Variables check its documentation in the PMAC Software Reference and set the Data Range attributes for the control to avoid problems with incorrect data entries. You can also set the action desired for out of range values. Feel free to use the numeric or boolean control of your choice.

The diagram for this VI demonstrates two ways of implementing I-Variable access. As a general principle, I-Variables should be read when the application begins and at anytime a different group is accessed. In this example, when the Motor Number is changed, the application logic generates a Boolean condition indicating this and re-initializes the panel controls. When an update button is clicked, the values contained in the controls are sent to PMAC.


Initialize the I-Vars when Motor Number changes
Set the I-Vars when the button is hit
True
I-Var GLng
ix30: Prop Gain
ix31: Deriv Gain
ix33: Integ Gain
I-Var SLng
Set - Hard
Generate an initialization condition when changing Motor Number
Initial value of -1 guarantees generation when starting VI
Motor Number
Dev Open
Set/Get = TRUE when setting - FALSE otherwise
PmacIVar... passes input value thru when setting. No need to test for Response Available
Set - Easy
ix30: Prop Gain - Easy
ix31: Deriv Gain - Easy
ix33: Integ Gain - Easy
IVar Long
Execute case when initializing or setting I-Vars
Stop

The use of PmacIVar as opposed to the PmacIVarSet and PmacIVarGet Vis, groups the controls into sub-units that are a little more manageable. They are used extensively in the ICVs introduced later.

PmacTutor6b – Accessing PMAC Memory

PMAC makes extensive use of the Motorola 56K memory mapped architecture. This includes various encoder registers, DAC and ADC values, digital I/O ports, etc. Details of PMAC's memory organization can be found in the PMAC Software Reference Manual and should be consulted when accessing memory.

The PmacMemory collection of VIs simplifies the access and manipulation of this architecture and its binary representation. LabVIEW numerical controls and indicators can be configured to display this information as either hex or decimal data independent of the integer representation of the data. Data is actually received from PMAC and sent to PMAC in this collection using ASCII hexadecimal strings.

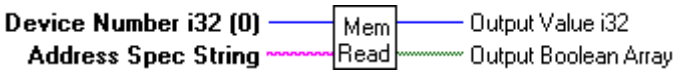
When defining a PMAC address to access the Address Spec String, input to the following VIs can be in either hexadecimal or decimal form. Both strings below access the same address:

```
Y:$C000
Y:49152
```

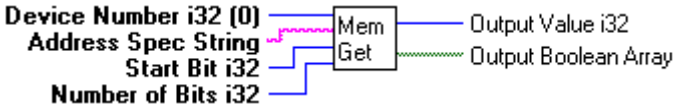
Reading Memory Data

There are two VIs to read and manipulate memory data in various forms. Remember that PMAC's integers are 24-bit words.

- PmacMemoryRead – Read a 24-bit quantity from the memory location specified by Address Spec String. For example, X:$002B. The result is output as both an i32 and a Boolean array.

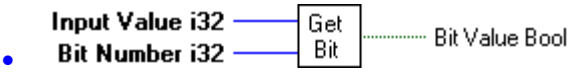


- PmacMemoryGet – Output Value is the value of the bit field defined by Start Bit and Number of Bits at the specified memory address. Output the field as both Output Value and Output Boolean Array.

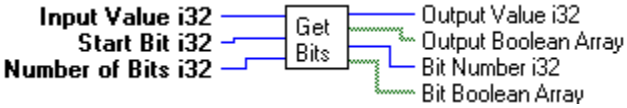


The data retrieved from PMAC can be manipulated using the following Vis:

- PmacMemoryGetBit – Bit Value is the bit at Bit Number in Input Value.



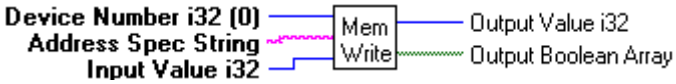
- PmacMemoryGetBits – Fetch the field defined by Start Bit and Number of Bits from Input Value. Return the field as Output Value i32, Output Boolean Array, and Bit Number (same as Output Value). Bit Boolean Array can be used with sets of radio buttons. If Output Value = 3, then Bit Boolean Array is 0, 0, 0, 1.



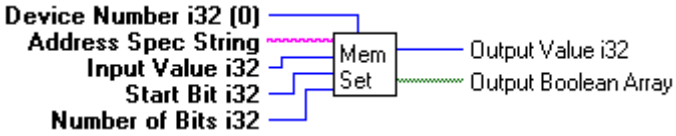
Writing Memory Data

There are two VIs to directly manipulate memory data in various forms. The data is first read from PMAC, modified, and then rewritten.

- PmacMemoryWrite – Write a 24-bit quantity (Input Value) to the memory location specified by Address Spec String (e.g. X:$002B). Pass Input Value to Output Value and Output Boolean Array.

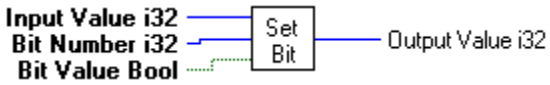


- PmacMemorySet – Write Input Value to a bit field defined by Start Bit and Number of Bits at the specified memory address. Output Value and Output Boolean Array are the value of the entire memory location with the new field.

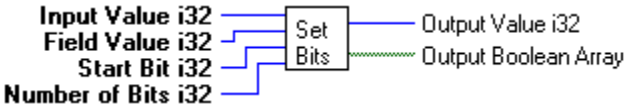


To implement these Vis, the following two VIs are used to manipulate the data:

- PmacMemorySetBit – Set Bit Number in Input Value using Bit Value. The new word is Output Value.



- PmacMemorySetBits – Insert Field Value into Input Value at the field defined by Start Bit and Number of Bits. Output Value is Input Value with Field Value inserted. Output Boolean Array is the Boolean representation of Output Value.



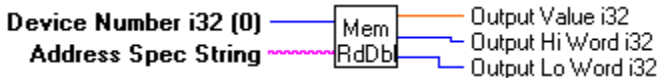
Reading and Writing 48 Bit Memory Data

Double word (48-bit) memory data is handled differently than single word (24-bit) data. PmacMemoryReadDbl and PmacMemoryWriteDbl provide two representations of the data - native LabVIEW double and two i32 integers one for the Hi X word and one for the Lo Y word. Do not attempt to access bits using logical bitwise operations such as Value and 32 on the double representation. Test them using logical comparison operations such as Value == 32. Bitwise operations on the Lo and Hi word are all right. Specifying addresses for double words must be done using the following notation:

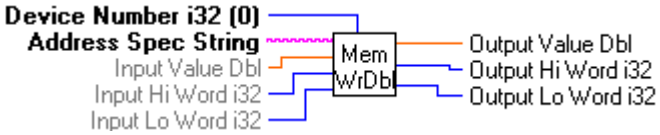
```
L:$002b
```

Specifying the address as `D:$002b` is not recognized by PMAC.

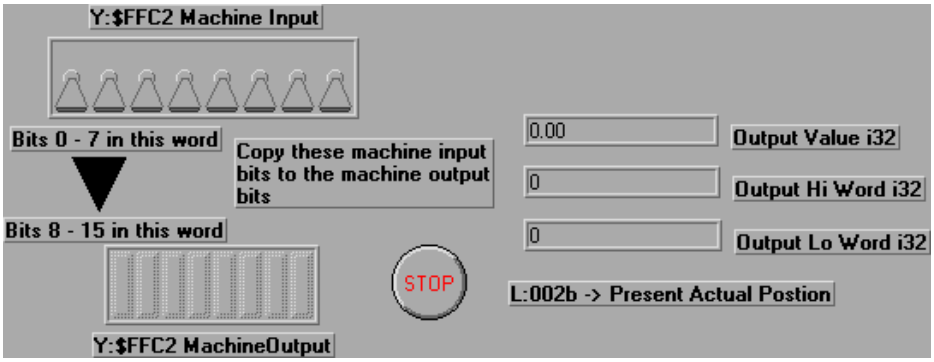
- PmacMemoryReadDbl - Read a 48-bit quantity from the memory location specified by Address Spec String. For example, L:$002B. The result is output as both a double and a hi-word and lo-word.



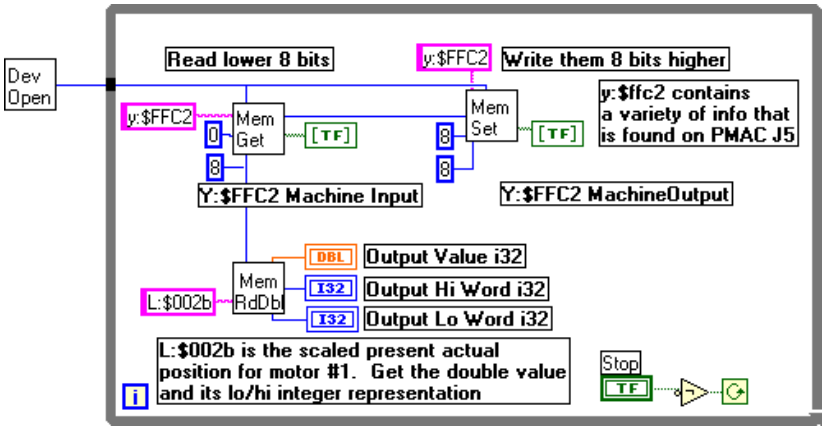
- PmacMemoryWriteDbl - Write a 48-bit quantity (Input Value) to the memory location specified by Address Spec String. For example, L:$002B. Input Value is copied to Output Value.



The panel for the exercise demonstrates the reading of a memory location containing the standard Machine Input at Y:$FFC2 to the standard Machine Output at Y:$FFC2. It also demonstrates accessing the 48-bit long word L:$002B that is the Present Actual Position for motor 1.



The diagram for this exercise demonstrates how the lower eight bits of Y:$FFC2 are written to the same memory location eight bits higher.



PMAC Panel Pro ICVs

The previous set of exercises introduced PMAC Panel Pro's Device, Communication, and Query/Response interfaces to PMAC. They handle the details of sending commands and requests to PMAC and converting basic responses into LabVIEW data formats. The exercises in this section introduce another level of PMAC Panel Pro capabilities that provide indicators, controls, and VIs for many of PMAC's most common on-line commands.

- Handle queries for motor, coordinate system, and global commands
- Define common indicator and control clusters for use on the panels
- Implement function VIs for the indicators and controls

Using these as is, and modifying others, allows the creation of great looking panels for applications quickly. PMAC Panel Pro's ICV collections are organized into five categories:

1. PmacAcc
2. PmacMotor
3. PmacMotors
4. PmacCoord
5. PmacGlobal

Each of these categories has several exercises to introduce its capabilities. There are similar examples in their respective sub-directories.

Each tutorial introduces an example and then selectively drills its way into supporting VIs. In doing so the developer will get a deeper understanding of PMAC Panel Pro internals so that potential limitations in the design can be addressed and its capabilities can be enhanced to suit specific requirements.

On-line Commands

PMAC provides a large selection of on-line commands for monitoring and control. Any of these commands can be sent to PMAC using the VIs already introduced. Not every command is supported or used by the ICVs introduced here. Some are rarely used. Some should really be used from Pewn32. Others are potentially dangerous (O100 turns a motor on 100%). Many would and some should never be used in a user application. A complete listing of available commands and their use can be found in the PMAC Software Reference Manual. The PMAC on-line commands used by PMAC Panel Pro VIs are listed here.

Global Commands

Addressing Mode Commands

&	Report currently addressed CS

Buffer Control Commands

CLOSE	Close an open program buffer
DELETE GATHER	Return gather buffer space
LIST PLC	List a PLC in memory
LIST PROG	List a program in memory
SIZE	Return available buffer space

Control-Character Commands

<CONTROL A>	Abort all programs and moves
<CONTROL D>	Disable all PLC programs
<CONTROL F>	Report following errors for all motors
<CONTROL K>	Kill all motors
<CONTROL P>	Report position of all motors
<CONTROL Q>	Quit all executing motion programs
<CONTROL R>	Begin execution of motion programs in all coordinate systems
<CONTROL S>	Step working motion programs in all coordinate systems
<CONTROL V>	Report velocity of all motors

General Global Commands

$$$	Full card reset
$$$* **	Reinitialize PMAC to factory default
???	Report global status words
SAVE	Save current configuration to NOVRAM

Global Variable Commands

I, P, Q, and M	Variable access in numerous ways.

PLC Control Commands

DISABLE PLC	Disable a PLC program
ENABLE PLC	Enable a PLC program

Register Access Commands

R[H]{address}	Read data from memory
W{address}	Write data to memory

Coordinate System Commands

Axis Definition Commands

#{constant}->	Query PMAC for motor definition in CS

Buffer Control Commands

LIST PC, PE	List program at Program Execution

General CS Commands

??	Report coordinate system status words

Program Control Commands

A	Abort program
B	Begin program
H	Hold program
Q	Quit program
R	Run program
S`	Step program

Motor Commands

General Motor Commands

$	Reset motor
HOME	Home
HOMEZ	Zero move home
K	Kill output

Jogging Commands

J+	Jog positive
J^{constant}	Jog relative to actual position
J/	Jog stop
J-	Jog negative
J={constant}	Jog to position
J=	Return to pre-jog

Reporting Commands

P	Report addressed motor position
V	Report addressed motor velocity
F	Report addressed motor following error
?	Report addressed motor status

PmacMotor ICVs

This series of exercises introduce the contents of the PmacMotor collection of ICVs. These allow the applications to add configuration, control, and monitoring for individual motors.

PmacTutor7 – Position, Velocity, Error, and Jogging

The most basic motion operations involve controlling or jogging motors under manual control and monitoring the position, velocity, and following error during the move.

Requesting and Formatting P, V, and E

PmacMotor has three VIs that request and format motor position, velocity, and following error. These require a Coordinate Specify Cluster input. This cluster is more of a data type than a cluster associated with a specific control. It is often assembled from controls in the application.

It is defined as:

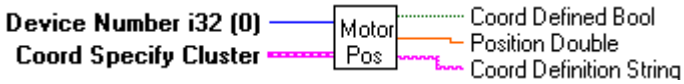
Coord Specify Cluster Cluster defining the motor, CS, and conversion state to be applied

- **Coord Number i32 (1-8) (1)** Coordinate number to use
- **Motor Number i32 (1-8) (1)** Motor Number to use
- **Convert Bool** Apply a conversion for the specified motor in the specified CS

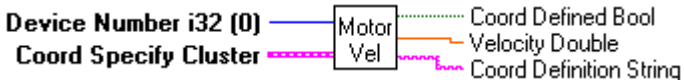
The VIs are:

- PmacMotorPosition – Query PMAC for Motor Number's position. PMAC reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bit 16 of Ix05 is 1 (handwheel offset mode), minus the master position register.

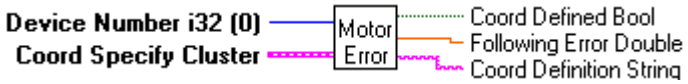
  Coord Specify Cluster specifies a motor within a CS and an attempt to convert motor position from encoder counts to CS units. If the motor is not defined in the CS, no conversion is applied. If the motor is defined and Convert is true, Coord Defined is true and position is scaled from encoder counts to CS units. Coord Definition is a string specifying position units as Encoder or the CS definition of the motor.



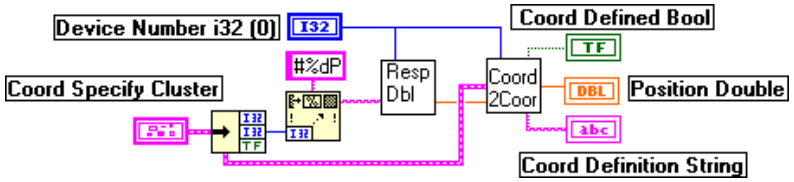
- PmacMotorVelocity – Query PMAC for Motor Number's present actual motor velocity, scaled in counts/servo cycle, rounded to the nearest tenth. The raw response reports the contents of the motor actual velocity register (divided by [Ix09*32]). This is converted to counts/msec by multiplying by 8,388,608 and dividing by the I10 default 3,713,707. If I10 is changed, modify this value in the diagram.



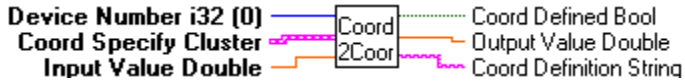
- PmacMotorError – Query PMAC for Motor Number's following error. Following error is the difference between motor desired and measured position at any instant. When the motor is open-loop (killed or enabled), following error does not exist and PMAC reports a value of zero.



The implementations of these VIs rely on the PmacCoord collection to convert values reported in motor encoder counts to coordinate system units. This capability is a fundamental component of PMAC Panel Pro. The diagram for PmacMotorPosition shows that these VIs format a command string to request the desired motor position and let PmacCoordMotor2Coord process the response.

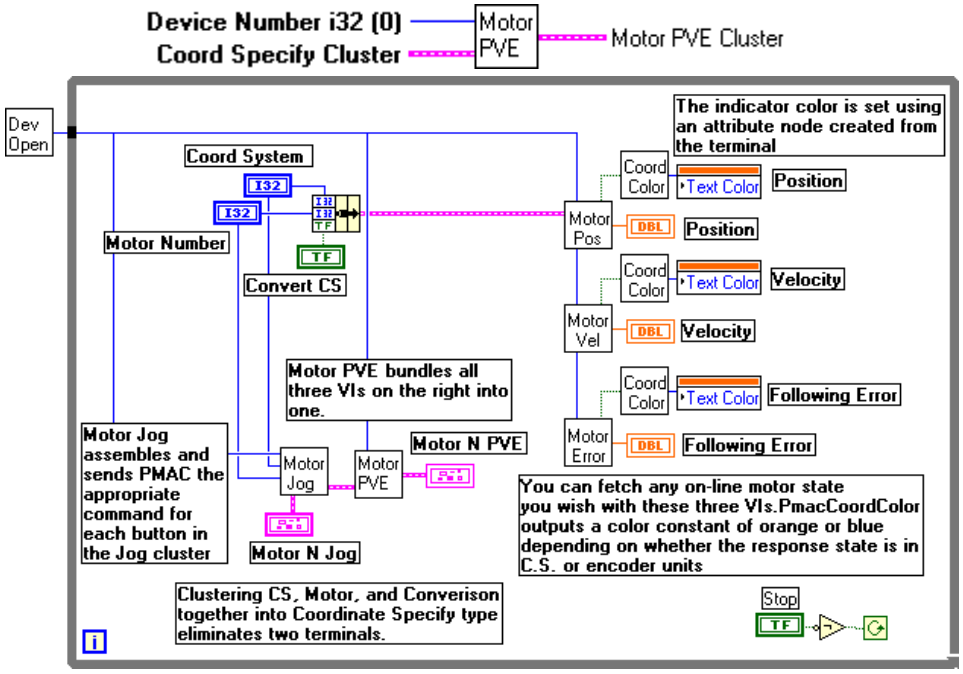


- PmacCoordMotor2Coord – Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from encoder counts to CS units. If the motor is not defined in the CS, no conversion is applied. If the motor is defined and Convert is true, Coord Defined is true, and Output Value is scaled from encoder counts to CS units. Coord Definition is a string specifying Output Value units as Encoder or the CS definition of the motor.

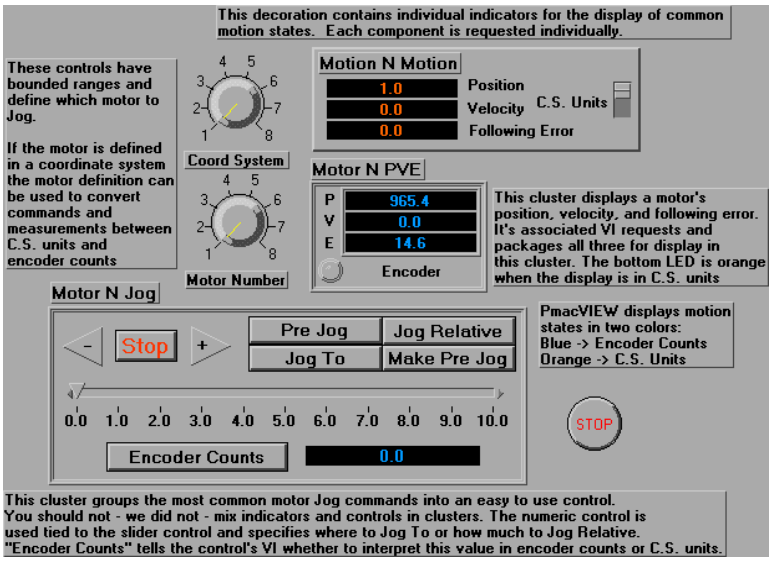


PmacMotorPVE is an extension of the position, velocity, and following error VIs that combine them into a single VI that can be used to drive the PmacMotorPVE indicator. The use of these VIs is shown in the diagram below.

- PmacMotorPVE – Query PMAC for the position, velocity, and following error for the Motor Number operating in Coord Number. Assemble the measurements into Motor PVE Cluster. If Convert is true, convert the measurements to CS units. Otherwise, leave them in encoder counts. See the documentation for PmacMotorPosition, PmacMotorVelocity, and PmacMotorError for details on how these individual values are produced.


Device Number i32 (0)
Coord Specify Cluster
Motor PVE
Motor PVE Cluster
Dev Open
Coord System
Motor Number
Convert CS
The indicator color is set using an attribute node created from the terminal
Coord Color
Text Color
Position
Motor Pos
Velocity
Motor Vel
Following Error
Motor Error
Motor PVE bundles all three VIs on the right into one.
Motor N PVE
Motor Jog assembles and sends PMAC the appropriate command for each button in the Jog cluster
Motor Jog
Motor N Jog
You can fetch any on-line motor state you wish with these three VIs.PmacCoordColor outputs a color constant of orange or blue depending on whether the response state is in C.S. or encoder units
Clustering CS, Motor, and Converison together into Coordinate Specify type eliminates two terminals.
Stop

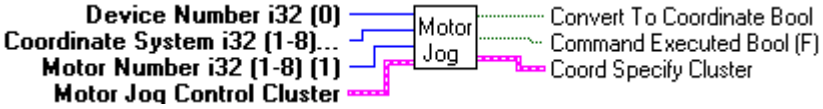
A Coordinate Specify Cluster is assembled from three controls on the panel. Each of the individual Position, Velocity, and Error VIs receives the cluster. The P, V, and E VI outputs are used to drive three individual numeric indicators. In this example, the indicators have an attribute node to set their color. The color for the text is provided by PmacCoordColor. The indicator color is orange if the reported value is in CS units and blue if it is in encoder counts. The panel for this exercise shows the indicators and the Boolean used to specify the units for P, V, and E displays.


This decoration contains individual indicators for the display of common motion states. Each component is requested individually.
These controls have bounded ranges and define which motor to Jog.
If the motor is defined in a coordinate system the motor definition can be used to convert commands and measurements between C.S. units and encoder counts
Coord System
Motor Number
Motor N Motion
1.0
Position
0.0
Velocity
C.S. Units
0.0
Following Error
Motor N PVE
P
965.4
V
0.0
E
14.6
Encoder
This cluster displays a motor's position, velocity, and following error. It's associated VI requests and packages all three for display in this cluster. The bottom LED is orange when the display is in C.S. units
Motor N Jog
Stop
Pre Jog
Jog Relative
Jog To
Make Pre Jog
PmacVIEW displays motion states in two colors:
Blue -> Encoder Counts
Orange -> C.S. Units
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0
STOP
Encoder Counts
0.0
This cluster groups the most common motor Jog commands into an easy to use control.
You should not - we did not - mix indicators and controls in clusters. The numeric control is used tied to the slider control and specifies where to Jog To or how much to Jog Relative.
"Encoder Counts" tells the control's VI whether to interpret this value in encoder counts or C.S. units.

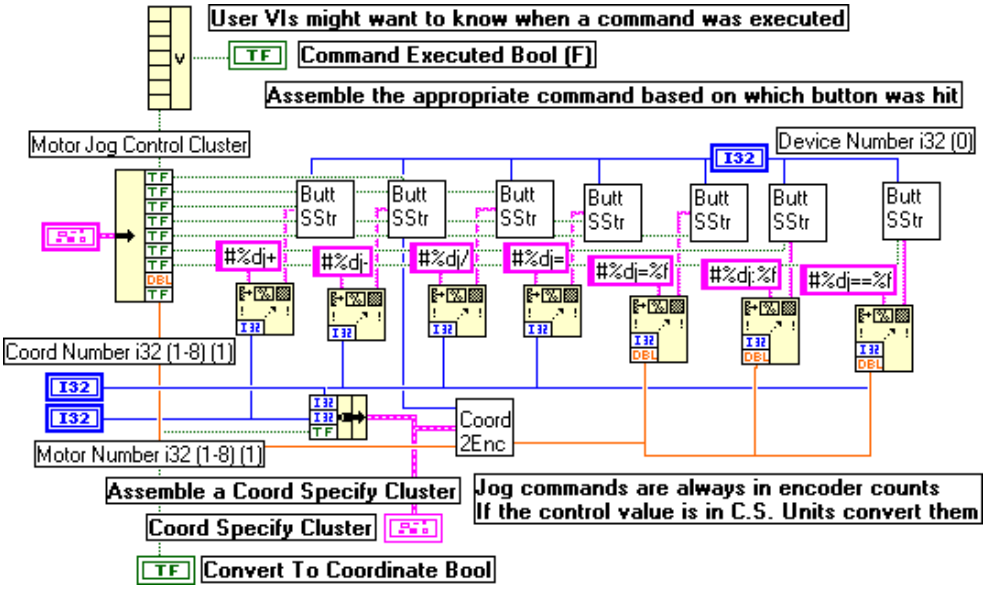
Generating On-Line Jog Commands

The front panel contains a cluster of controls defined by PmacMotorJogControl to group the most commonly used jog operations into a single panel item. This cluster must be used with its associated function VI to generate jog commands for PMAC.

- PmacMotorJogControl – Generate PMAC on-line commands for controlling jogging Motor Number. Command Executed Bool is true when any button is clicked in Motor Jog Cluster. The value in the numeric control specifies the position Jog To and Jog Relative jog the motor to. This value is interpreted as either Encoder Counts (Default) or Coordinate Units in Coord Number as specified on the button. The button state is provided as the output Convert to Coordinate. This VI builds a Coord Specify Cluster using the various inputs to simplify the interface to PmacMotorPVE and other PMAC Panel Pro ICVs.



When buttons in the cluster are clicked, the appropriate on-line command is assembled and sent to PMAC. The diagram for this VI illustrates the general architecture PMAC Panel Pro uses to generate on-line commands from control clusters.


User VIs might want to know when a command was executed
Command Executed Bool (F)
Assemble the appropriate command based on which button was hit
Motor Jog Control Cluster
Device Number i32 (0)
Butt SStr
#%dj+
#%dj-
#%dj^
#%dj=
#%dj=%f
#%dj:%f
#%dj==%f
Coord Number i32 (1-8) (1)
Coord 2Enc
Motor Number i32 (1-8) (1)
Assemble a Coord Specify Cluster
Jog commands are always in encoder counts
If the control value is in C.S. Units convert them
Coord Specify Cluster
Convert To Coordinate Bool

Note:

If generating Motor or C.S. specific commands, send the Motor or C.S. number along with the command.

There are a few things to note about the organization of this VI. Unbundled command buttons are used in conjunction with a string format VI and PmacButtSendStr to create and send an appropriate command to PMAC. The position parameter required by some commands is converted into encoder units if the actual control value is in C.S. units. Using this architecture eliminates many case structures and allows adding or deleting commands as required.

Using PmacMotorJogControl and its cluster with PmacMotorPVE and its cluster, motor jogging and position monitoring can be added quickly on any panel.

Control Clusters and Local Variables

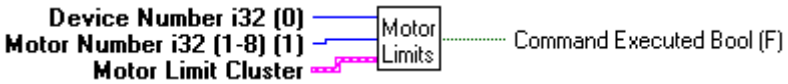
It has already been noted that clusters should contain either indicators or controls but not both. When a VI reads the state of a latched Boolean control, the control is reset. Hence, if there were two users of the Boolean state the second one to read the state would get the wrong answer. A corollary to this rule is that not to use local variable copies of control clusters that contain latched Boolean controls.

PmacTutor8 – Motor Control with Status Monitoring

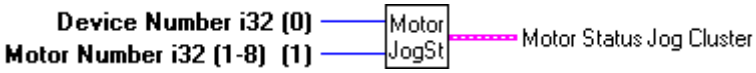
This exercise introduces another PmacMotor control and several indicator clusters for displaying motor status. As with the PmacMotorJogControl, status clusters require an associated VI to query PMAC for the required information and assemble it into a useful form.

This is the first exercise to actually generate an application that can be used to exercise PMAC. The panel contains PmacMotorJogControl and PmacMotorPVE already introduced. In addition, there is a PmacMotorLimitControl and two new status clusters. The VIs for the new panel are:

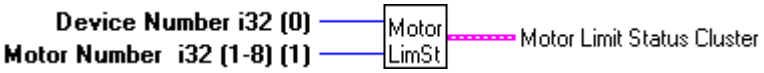
- PmacMotorLimitControl – Generate PMAC on-line commands for controlling move limits and operation for Motor Number. Command Executed Bool is true when any button is clicked in Motor Limit Cluster.

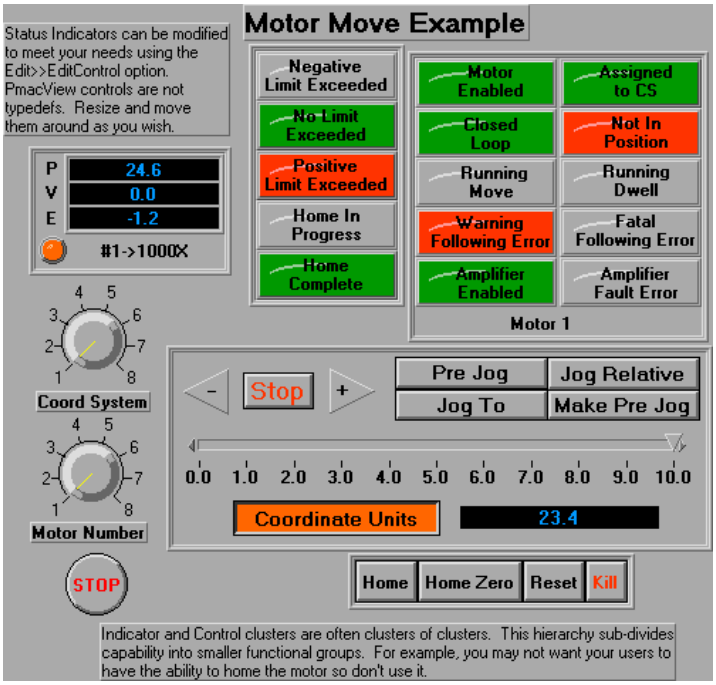


- PmacMotorStatJog – Create a status indicator cluster for the PmacMotorStatJog indicator containing the status for Motor Number.



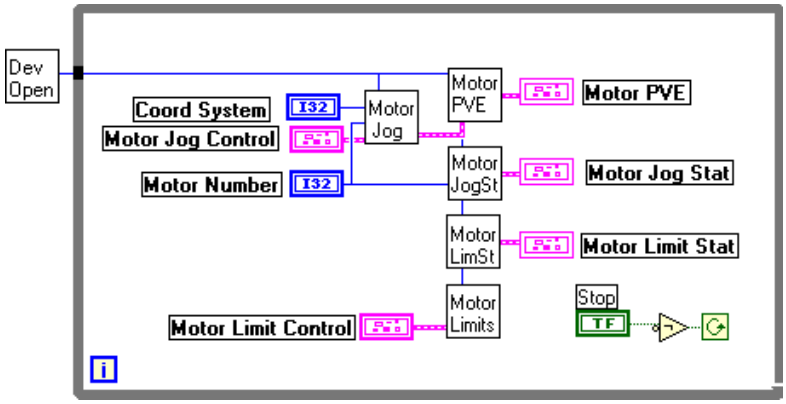
- PmacMotorStatLimit – Create a status indicator cluster for the PmacMotorStatLimit indicator containing the status for Motor Number.





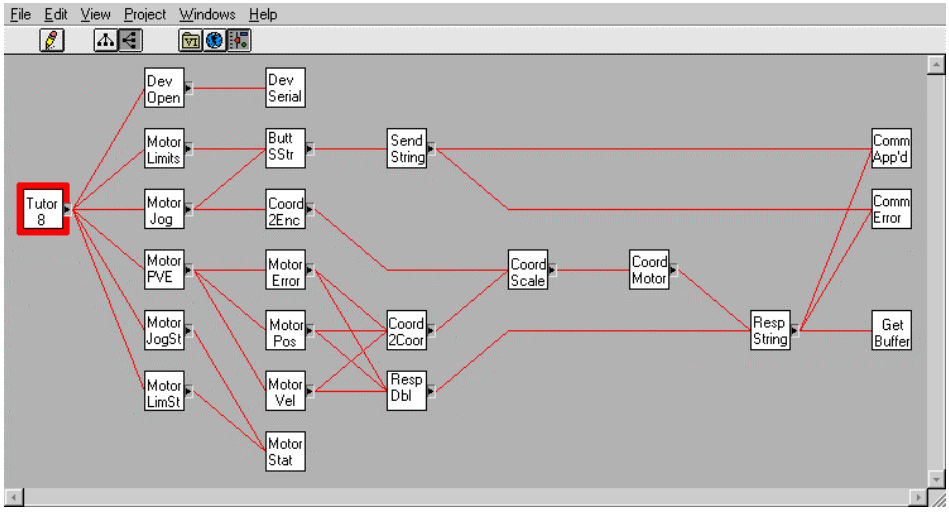
This panel, captured while actually running, demonstrates how useful these new capabilities are. The jog control indicates a move to a position of 23.4 defined in Coordinate Units, not motor encoder counts. The PVE indicator has an orange LED indicating that the displayed position, velocity, and following errors are in CS units. Furthermore, because the Coord System dial is set to 1, motor 1 is defined as #1->1000X in CS 1. The status indicators show that Motor 1 is assigned to a CS, enabled, and in closed loop mode. Furthermore, that it is Not In Position and there is a Warning Following Error. Note that the position in the Jog Control is specified as 23.4 CS Units and the PVE indicator shows an actual position of 24.6 CS Units. If actually running this exercise, turn on Help by selecting **Help»Show Help**. As the cursor moves over the various indicator and control items in the clusters, detailed help for each panel item is displayed.

The diagram for this panel is simple because PMAC Panel Pro handles all the details. There are two control clusters, one PVE cluster, and two status clusters each with its associated function VI. Add two dials and a Jog application has been created. Note that the Coord Specify Cluster required by PmacMotorPVE is constructed by PmacMotorJog.



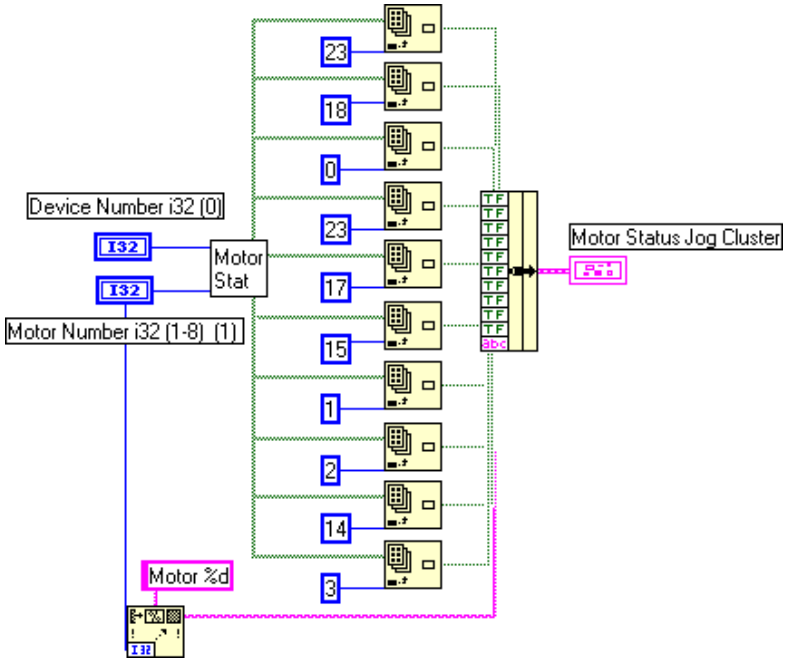
Hierarchical Encapsulation

PMAC Panel Pro attempts to break panel clusters and function VIs into manageable chunks that group functionality. Using this approach, piece together those items needed to build an application. Each VI builds on top of the capabilities provided by still lower levels until almost everything funnels through PmacCommSendStr and PmacCommRespStr. In using proper program design, the result is easier to maintain and modify for individual purposes. The VI Hierarchy for this exercise is shown here to illustrate this point.



Accessing Status Bits

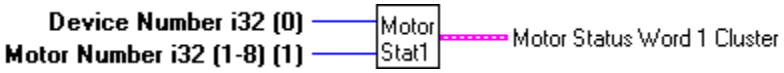
To illustrate how PMAC Panel Pro handles status information, the diagram for PmacMotorStatJog is shown here. The VI calls the PmacMotorStat VI covered in PmacTutor5 and the Boolean array for each status word is indexed to get the desired bit. The individual bits are assembled into a cluster for use by the indicator. Notice that a string is created indicating the associated motor in the status cluster. It updates the indicator cluster automatically. To remove this, simply eliminate it from the cluster and modify the panel cluster.

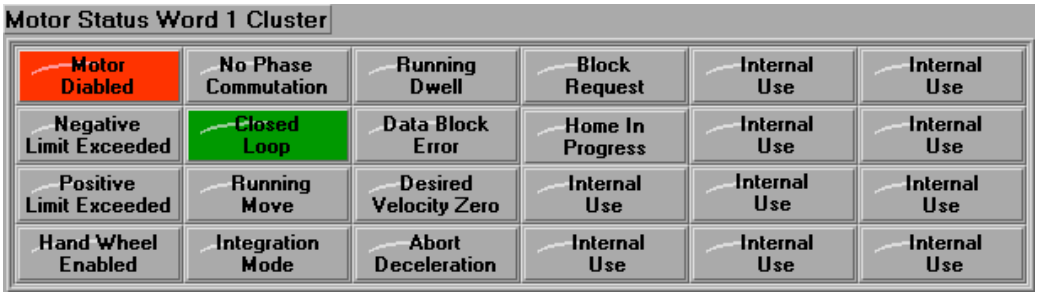


Motor Status VIs

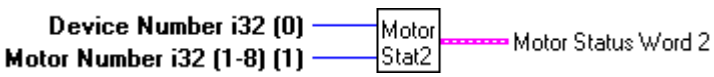
For completeness, PMAC Panel Pro provides indicator/VI pairs to monitor all bits in both motor status words. Copy the individual cluster items into custom clusters to avoid creating them from scratch. Each cluster LED has been carefully constructed to include detailed help, labeling, and Boolean text. The icons along with the panel clusters are

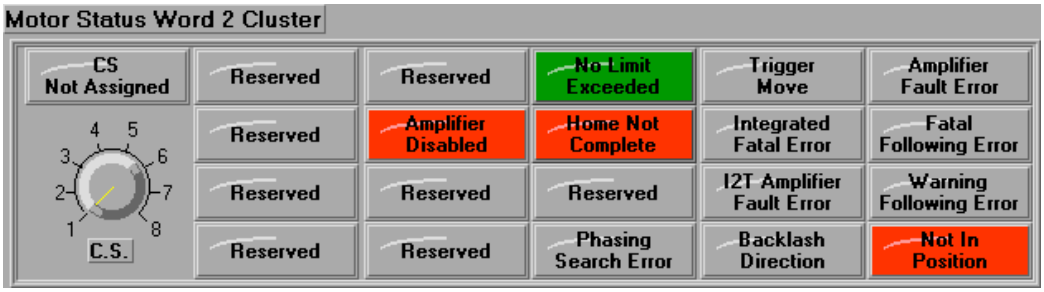
- PmacMotorStat1 – Create a status indicator cluster for the PmacMotorStat1 indicator containing the status for Motor Number.





- PmacMotorStat2 – Create a status indicator cluster for the PmacMotorStat2 indicator containing the status for Motor Number.





These clusters are organized to reflect the 4-bit organization of the status words. The first column is for the first four bits (23-20), the second column for the second four (19-16), etc. Note that some bits are not defined. They are reserved or for internal use. Also, note that for motor status word, two 3-bits are interpreted as the coordinate system to which the motor is assigned.

Status Indicator Colors

Human factor considerations play a major role in how colors are assigned to an application's status indicators. Is green good? Is it true? If green is true and red is false, then an Amplifier Fault is green. Setting up the definitions can be very confusing.

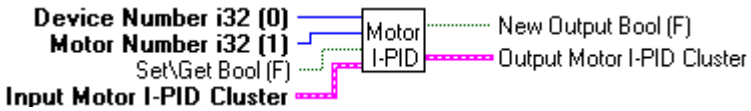
In PMAC Panel Pro, status indicators attempt to convey a generally useful meaning by the LED's color and text. To clarify this a few examples are covered in more detail. Feel free to change colors and text to reflect the intent within the application.

- Following Errors: These are red when there is an error. For these status bits that means the bit is true. When the bit is false, the indicators are gray - there is no error. Gray can generally be interpreted as not true or not dangerous. The text in the indicator says what the indicator is Warning Following Error either way.
- In Position: This indicator is red when the motor is not in position (the bit is false). Its text says - Not In Position. The indicator is green and says In Position when the motor is in position (the bit is true). Imagine a situation where motor not in position means its moving, therefore the program is running, and that is good.
- Home in Progress: This is gray when there is no Home in Progress. This does not mean that the motor has been homed. When the motor is homing, the indicator is green. It may well be that this should be red when the motor is homing.

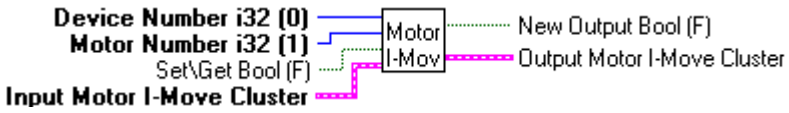
PmacTutor9 - Motor I-Variable Configuration

In PmacTutor6, the general architecture for developing supervisory VIs for modifying individual I-Variables was introduced. In this exercise, four VIs to encapsulate the most common motor I-Variables are introduced. This type of hierarchical I-Variable architecture is used throughout PMAC Panel Pro.

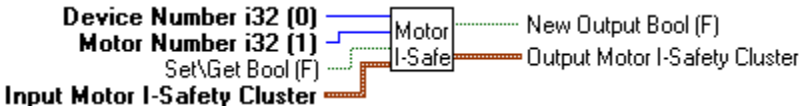
- PmacMotorIVarPID – Follow PMAC Panel Pro's standard I-Variable VI architecture. When Set/Get is true, the PID I-Variables for the specified Motor Number are set. Otherwise they are fetched from PMAC and provided by Output Motor I-PID Cluster with New Output true. Set/Get is not required and defaults to a Get operation.



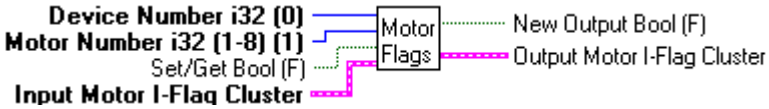
- PmacMotorIVarMove – Follow PMAC Panel Pro's standard I-Variable VI architecture. When Set/Get is true, the movement I-Variables for the specified Motor Number are set. Otherwise they are fetched from PMAC and provided by Output Motor I-Move Cluster with New Output true. Set/Get is not required and defaults to a Get operation.



- PmacMotorIVarSafety – Follow PMAC Panel Pro's standard I-Variable VI architecture. When Set/Get is true, the safety I-Variables for the specified Motor Number are set. Otherwise they are fetched from PMAC and provided by Output Motor I-Safety Cluster with New Output true. Set/Get is not required and defaults to a Get operation.

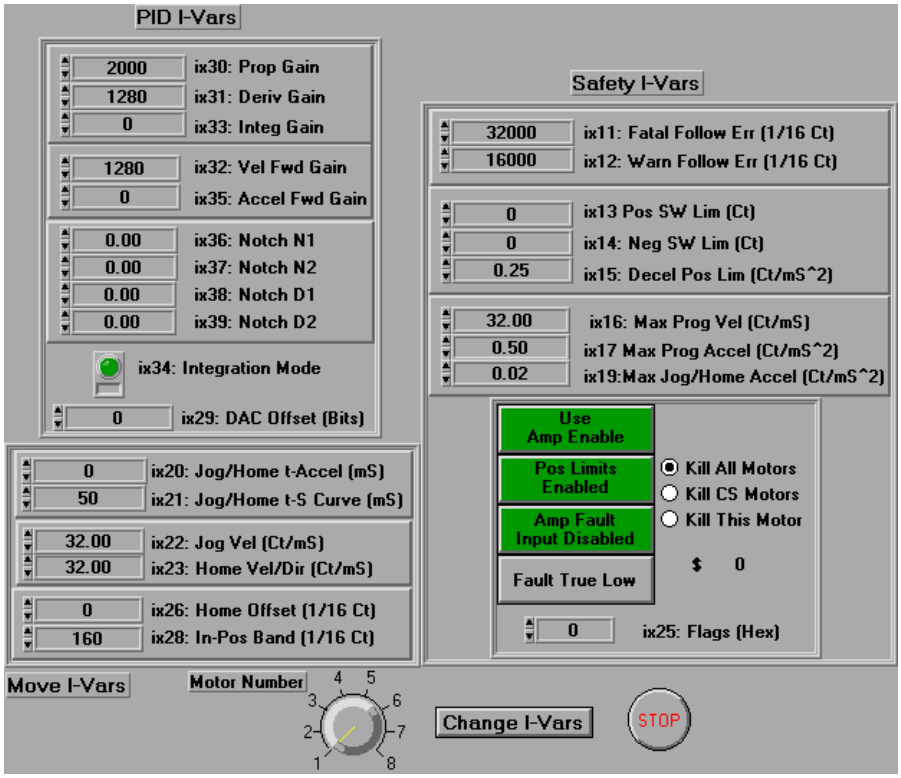


- PmacMotorIVarFlag – Follow PMAC Panel Pro's standard I-Variable VI architecture. When Set/Get is true, the encoder flag I-Variable Ix25 for the specified Coord Number are set. Otherwise they are fetched from PMAC and provided by Output Coord I-Flag Cluster with New Output true. Set/Get is not required and defaults to a Get operation.

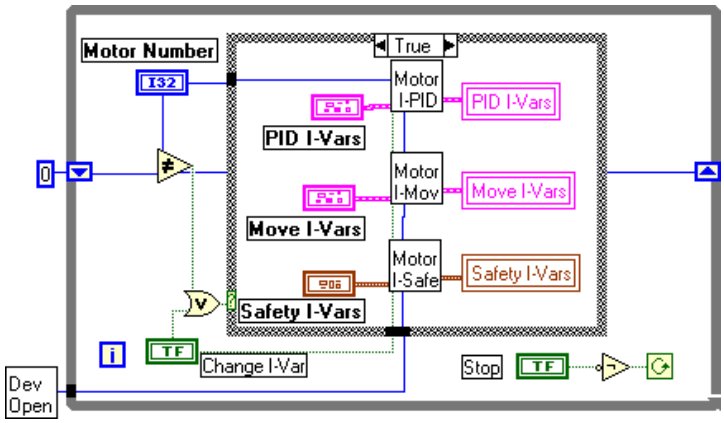


The panel for this exercise shows the three main PmacMotorIVar clusters. PmacMotorIVarFlag is actually a sub-cluster in PmacMotorIVarSafety.

To change the I-Variables, click the **Change I-Vars** button. Each item in the cluster has a full description that is accessible using **Help»Show Help**. Because each item in the clusters have been defined as referencing a specific I-Variable, the item units and appropriate data ranges for each item have been specified.



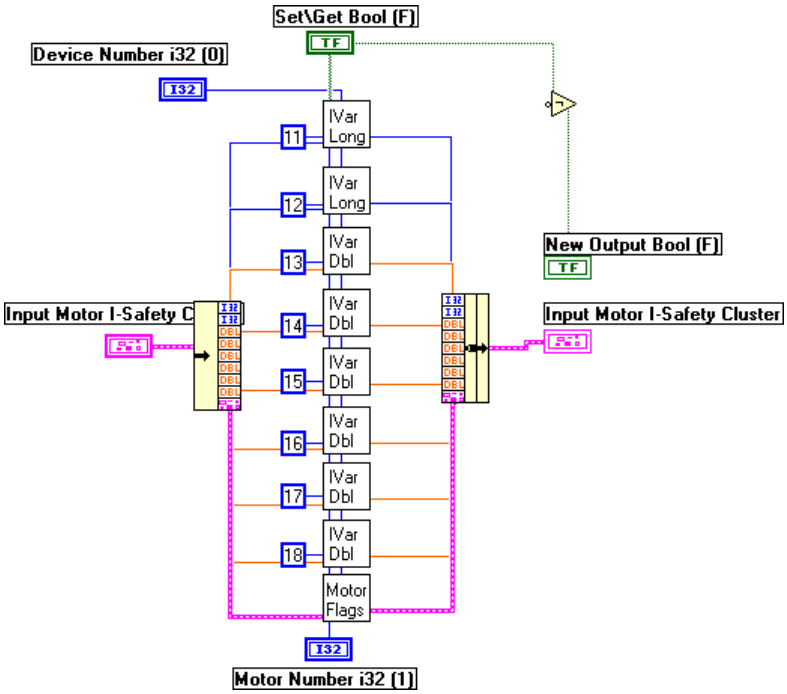
The diagram for the exercise demonstrates why using these ICVs makes life easier. The Boolean criteria for executing the case are the same as in PmacTutor6. It is executed whenever Motor Number is changed or the button is clicked. When the case is executed because of a change in Motor Number, the VIs perform a get operation and refresh the cluster contents with the configuration of the new motor.



The observant reader will note that the VI makes use of local copies of the panel control/indicator clusters. Those clusters that contain Boolean controls such as PmacMotorIVarPID and PmacMotorIVarFlag have their Booleans set for pressed mechanical action not latched action. They are not used to initiate a command or action. This does not violate the basic caution on using local copies of control clusters noted in this document in several places.

Grouping Multiple I-Variables

It should now be obvious why the PmacIVar VIs require an I-Var Set Number. It allows them to be grouped by motor and/or coordinate system. To accomplish this, it is only necessary to extend the concept to a slightly higher level. The diagram for PmacMotorIVarSafety is shown here. Note that when one I-Variable is set, they are all set whether or not they have changed. Therefore, the contents of the cluster should be refreshed by a Get operation prior to changing individual items and performing a Set operation. Controls and indicators for the panels should have the appropriate type and range defined to prevent inadvertent user inputs.



PmacMotors ICVs

This series of exercises introduce the PmacMotors collection of ICVs. These allow the applications to monitor and plot the motion of multiple motors.

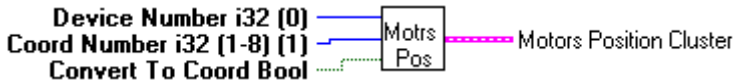
PmacTutor10 - Requesting and Plotting Motor Motion

This exercise introduces a number of indicators, controls, and VIs for requesting the motion of all motors. The data can be displayed on a PMAC Panel Pro cluster indicator, plotted in a strip chart, or analyzed using LabVIEW's extensive analysis capabilities. VIs for setting plot legends and selecting which motors to plot can be used to create flexible interfaces.

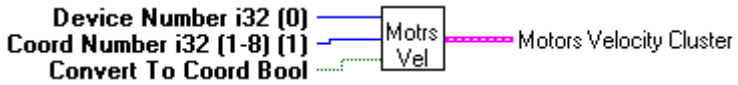
The primary query VIs in the collection request PMAC position, velocity, and following errors for all motors. They are not based on their counterparts in PmacMotor.

- PmacMotorsPositions – Query PMAC for the positions for all motors. PMAC reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bit 16 of Ix05 is 1 (handwheel offset mode), minus the master position register.

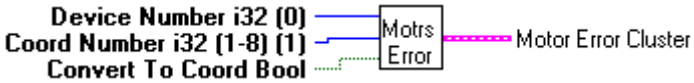
Assemble the measurements into PmacMotorsPVE Cluster. If Convert to Coord is true, convert the measurements to CS units for those motors defined in the CS. Otherwise, leave them in encoder counts.



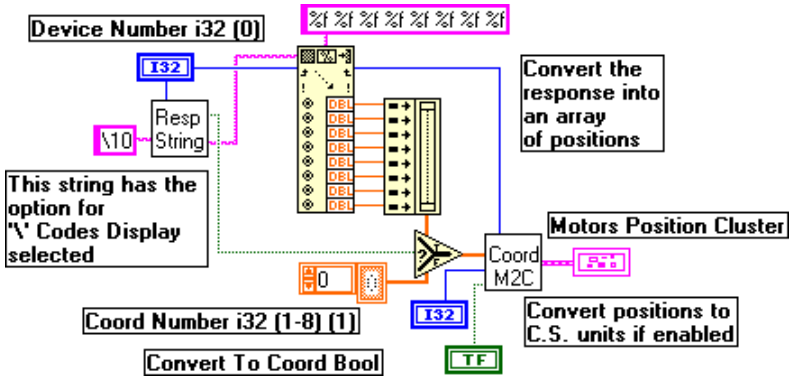
- PmacMotorsVelocities – Query PMAC for all motor's present actual motor velocity, scaled in counts/servo cycle, rounded to the nearest tenth. The raw response reports the contents of the motor actual velocity register (divided by [Ix09*32]). This is converted to counts/msec by multiplying by 8,388,608 and dividing by the I10 default 3,713,707. If I10 is changed, modify this value in the diagram.



- PmacMotorsErrors – Query PMAC for the following errors for all motors. Following error is the difference between a motor's desired and measured position at any instant. When a motor is open-loop (killed or enabled), following error does not exist and PMAC reports a value of 0.



The diagram for PmacMotorsPositions shows that a command string is sent requesting the position. This command is a control code and requires the proper option be set for the string constant. The response will have as many positions as there are motors in PMAC. The responses are converted into an array and processed by PmacCoordMotors2Coord which assembles the results into a cluster for display on a PmacMotorsPVE cluster. This is different from a PmacMotorPVE cluster.



The PmacMotorsPVE cluster is comprised of an array of values, a Boolean array to indicate which motors have been converted to CS units and a text label for the cluster indicating which coordinate system the display is using for those motors displayed in C.S. units.

Motors PVE Cluster - The indicator cluster displays an array of values for all PMAC motors. The array may be positions, velocities, or following errors. The array of Boolean indicators indicate which values are in CS units. The caption specifies the displayed values as being in encoder counts or a specific CS.

See the documentation for PmacMotor's Position, PmacMotor'sVelocity, and PmacMotor'sError for details on how these individual values are produced.

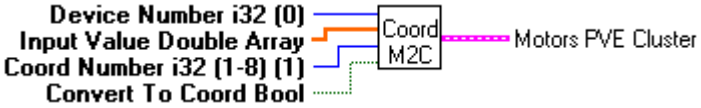
Motor Value Array - Array of numerics for positions, velocities, or following errors for each motor. See the documentation for PmacMotors Position, PmacMotorsVelocity, and PmacMotorsError for details on how these individual values are produced.

C.S. Defined - Array of Booleans indicating which motors are displayed in CS Units.

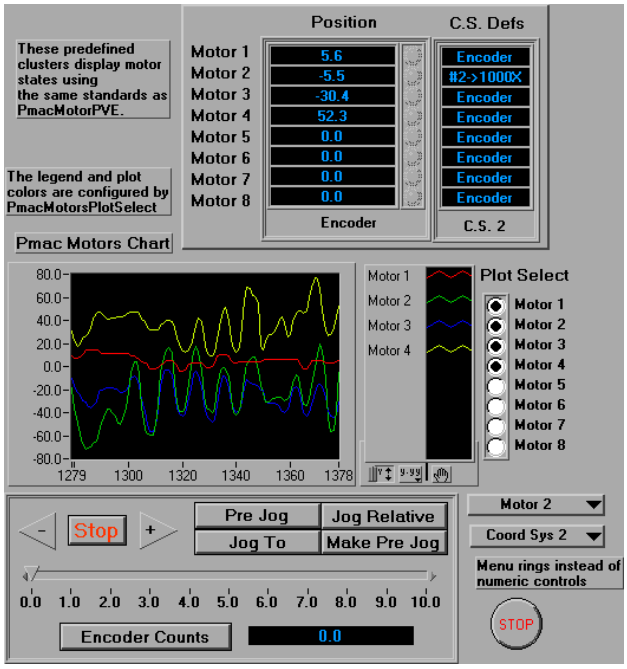
C.S. Applied - Caption indicating the currently addressed coordinate system or that the displayed values are in Encode Counts.

The conversion of motor states from encoder counts to CS Units operates similar to PmacCoordMotorTo Coord introduced in PmacTutor7.

- PmacCoordMotorsToCoord - Generate an indicator cluster for PmacMotorsPVE. Input Value Double is an array of positions, velocities, or following errors from VIs in the PmacMotors collection. If Convert To Coord is true, fetch the CS definitions for the motors specified in Coord Number and scale them to CS units. Motors not defined in Coord Number are not scaled.



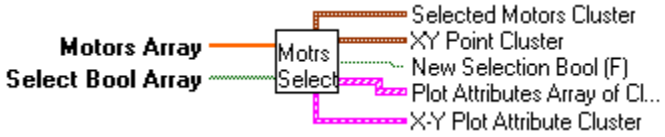
The panel for the exercise uses the familiar jog control with LabVIEW menu rings for selecting the motor number and coordinate system number. The PmacMotorsPVE cluster and a CS definition cluster display the motor positions and motor definitions in the addressed CS. The plot is a standard strip chart with Auto Scaling on the Y-Axis. To plot one or more motors, click the appropriate radio button. The colors for the plots and the legend are updated automatically.



Use the motor and CS menu rings to select motors to jog and coordinate systems to convert positions to CS units. If there are motion programs running on PMAC while this VI is executing, the position data plotted and displayed by this example will be from those motion programs. In either case, motor positions are displayed in real-time in the PmacMotorsPVE cluster and the strip chart.

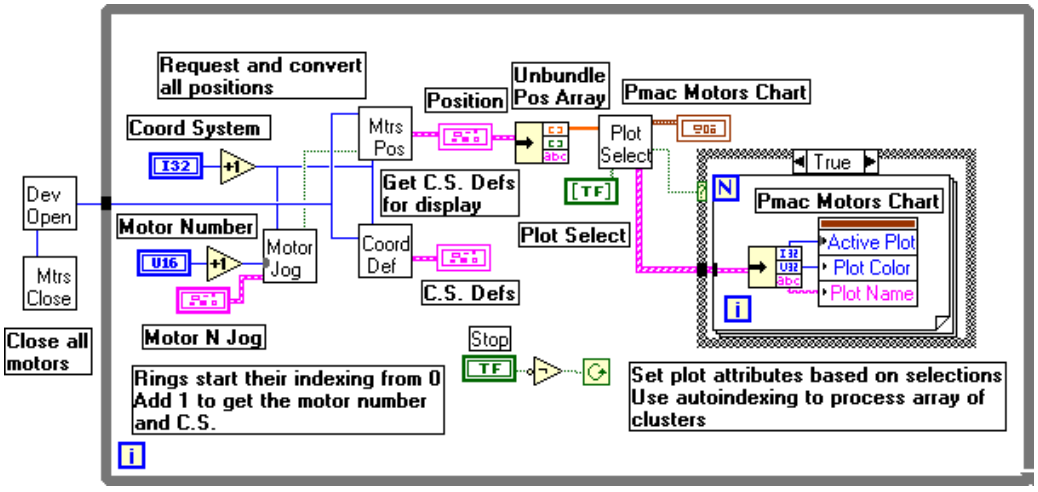
To implement this exercise, PmacMotorJogControl and PmacMotorsPositions are used. The CS definitions are retrieved using PmacCoordDef covered in a later exercise. The PmacMotorsPVE cluster generated by PmacMotorsPositions is unbundled and the array of values is passed to the PmacMotorsPlotSelect VI.

- PmacMotorsPlotSelect – Motors Array contains positions, velocities, or following errors for all motors on your PMAC. Select Bool Array defines which motors are copied into Selected Motors Cluster for plotting on LabVIEW strip charts. XY Point Cluster contains two values for X-Y plotting. New Selection is true when Select Bool changes and indicates the Plot Attributes Array of Clusters and X-Y Plot Attribute Cluster contain new information for updating plot legend attributes.



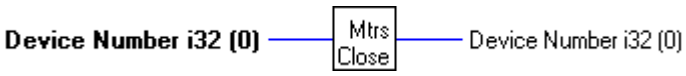
Setting the plot attributes is done in the case structure in the lower right of the diagram below. Plot Attributes Array of Clusters contains an Active Plot index, Plot Color, and Plot Name as supplied by PmacMotorsPlotSelect. When New Selection is true, the case is executed and the array is auto-indexed. The cluster of attributes is unbundled and used to set the three attributes for the plot.

One important thing to note is that Motor Number and Coord System are specified using LabVIEW rings. These controls start their indexing from 0 not 1. Hence, to use them to specify legitimate motors and coordinate systems they must be incremented by one. The range on the ring can be changed from 0-7 to 1-8. However, this leaves item zero in the ring empty.

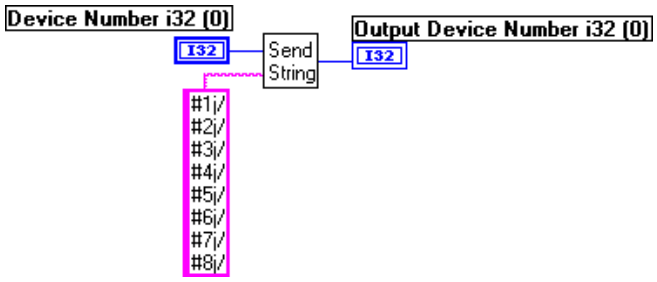


When developing the applications, create custom simple VIs to do common things. PmacMotorsClose is an example of this. The VI and its diagram are shown here. If PMAC is being send simple commands repeatedly in the application, begin creating a custom set of useful VIs.

- PmacMotorsClose – Close all PMAC motor loops.



The important thing to note about this VI is that eight commands are executed by PMAC. A **<CR>** separates the individual commands in the command string.



PmacGlobal ICVs

This series of exercises introduce the PmacGlobal collection of ICVs. These control, monitor, and configure PMAC's global characteristics.

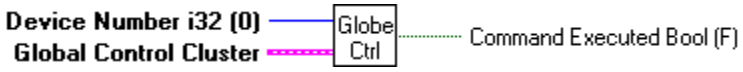
PmacTutor11 – Configuring PMAC's Global State

This exercise introduces a number of indicators, controls, and VIs for controlling and monitoring PMAC's general operational characteristics. This is done using status indicators, a simple control cluster for sending PMAC commands, and a few clusters for configuring I-Variables. In general, these capabilities are used for supervisory purposes and not exposed to general users.

The architecture for PmacGlobal ICVs follows those already introduced. The basics are introduced with these six VIs and their cousins.

- PmacGlobalBufferSize – Monitor and control PMAC buffer space during system development. Buffers Open is true if an open prog, open rotary, or open PLC command has been executed and the corresponding buffer has not been closed yet. Available Buffer Memory specifies how much buffer space PMAC has left for gathering and programs. A define gather command reserves all available buffer space. If Close Buffers is true, the gather buffer is deleted and a close command is sent to PMAC.

- PmacGlobalControl – Generate PMAC on-line commands for controlling PMAC program execution and save state. Command Executed Bool is true when any button is clicked in Global Control Cluster.

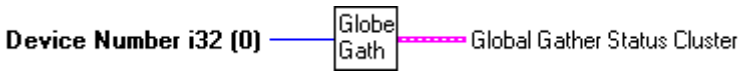


- PmacGlobalIVarComm – Follow PMAC Panel Pro's standard I-Variable VI architecture. When Set/Get is true, the global communication I-Variables are set. Otherwise they are fetched from PMAC and provided by Output Global I-Comm Cluster with New Output true. Set/Get is not required and defaults to a Get operation.

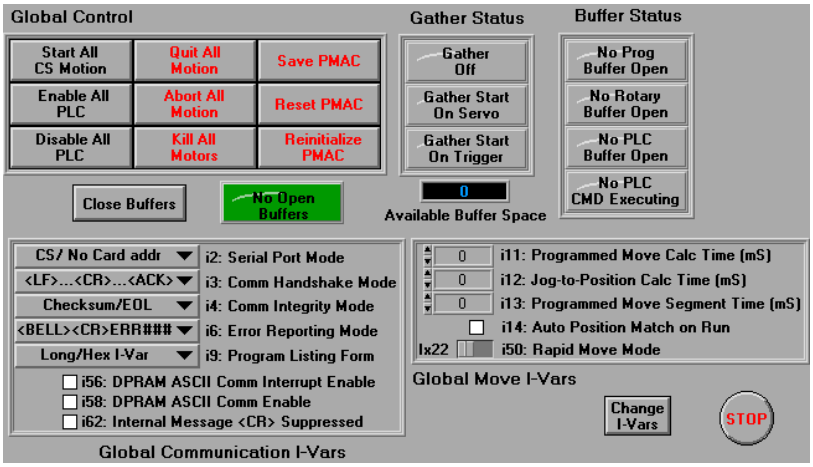
- PmacGlobalIVarMove – Follow PMAC Panel Pro's standard I-Variable VI architecture. When Set/Get is true, the global movement I-Variables are set. Otherwise they are fetched from PMAC and provided by Output Global I-Move Cluster with New Output true. Set/Get is not required and defaults to a Get operation.

- PmacGlobalStatBuffer – Create a status indicator cluster for the PmacGlobalStatBuffer indicator containing PMAC's global buffer status.

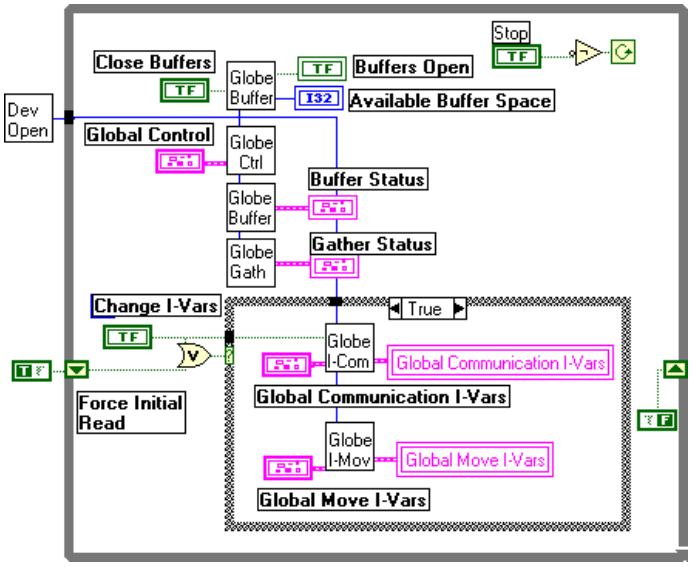
- PmacGlobalStatGather – Create a status indicator cluster for the PmacGlobalStatGather indicator containing PMAC's global gather status.



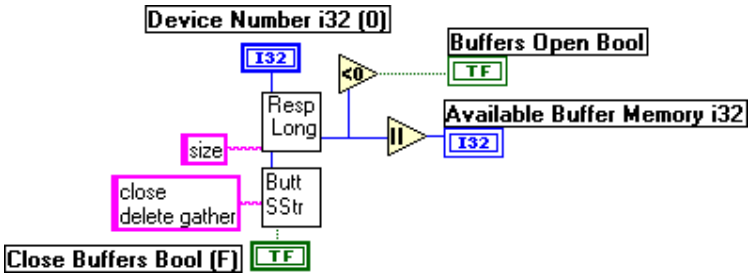
The panel for this exercise is the basis for one of PMAC Panel Pro's terminal tools. At the top is the PmacGlobalControl cluster. Use this to generate global commands for PLC and program control. During development, use the **Abort All Motion** button. The **Save**, **Reset**, and **Reinitialize** buttons allow PMAC's configuration to be saved to its onboard battery-backed RAM or flash memory. This way PMAC will always boot with the proper program and configuration information for the application. The two status indicator clusters might be useful during development, as will the ability to monitor PMAC's available buffer space. The two I-Variable clusters are useful when configuring communication or require specific program execution characteristics. For example, later exercises execute multi-axis motion using circular interpolation. This requires I13 > 0.



The diagram for this exercise should begin looking familiar. Most of the work is contained in the provided PMAC Panel Pro ICVs. The control cluster provides the input for PmacGlobalControl. Status VIs process requests for PMAC global status and create appropriate clusters. The architecture for configuring the I-Variables is slightly different from that used to configure motors. Motor Number equivalent cannot be changed. Instead, a shift register is initially set to true to force an I-Variable read and set false for every other iteration.



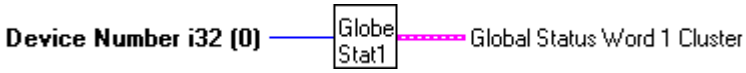
PmacGlobalBufferSize provides two capabilities. The VI queries PMAC for available buffer size and parses the state of the buffers and their size. When Close Buffers is true, PMAC executes a command to close the open buffers and delete the gather buffer. Generally, an open gather buffer prevents the download of any new program or modification of any existing program.



Global Status ICVs

In typical fashion, the PMAC Panel Pro global collection of ICVs provides a pair of indicator clusters and associated VIs to monitor all global status bits. The icons for the VIs and the indicator clusters are shown below. Unlike the PmacMotor status word clusters, these clusters have many status bits that are reserved and for internal use.

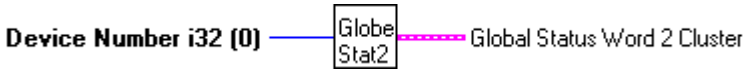
- PmacGlobalStatWord1 – Create a status indicator cluster for the PmacGlobalStat1 indicator containing PMAC's global status.



Real-Time Interrupt Active	Data Gather Function On	Reserved	DPRAM Error	Internal Use	Reserved
Real-Time Interrupt Re-ent	Data Gather Start on Servo	Leadscrew Comp On	EAROM Error	TWS Variable Parity Error	Reserved
Servo Active	Data Gather Start on Trigger	Any Memory Checksum Error	Internal Use	MACRO Aux Comm Error	All Cards Addressed
Servo Error	Reserved	PROM Checksum Error	Internal Use	Reserved	This Card Addressed

Global Status Word 1 Cluster

- PmacGlobalStatWord2 – Create a status indicator cluster for the PmacGlobalStat2 indicator containing PMAC's global status.



Internal Use	Motion Buffer Open	VME Comm Mode	Fixed Buffer Full	Reserved	Reserved
Host Comm Mode	Rotary Buffer Open	Internal Use	Internal Use	Reserved	Reserved
Internal Use	PLC Buffer Open	Internal Use	Internal Use	Reserved	Reserved
Internal Use	PLC Command	Internal Use	Internal Use	Reserved	Reserved

Global Status Word 2 Cluster

PmacCoord ICVs

This series of exercises introduce the PmacCoord collection of ICVs. Coordinate systems organize motors into familiar engineering measurement systems in which motion programs execute. They define the scaling between motor encoder counts and engineering units such as inches, centimeters, degrees, or radians. They can also define coupling between multiple motors and a single coordinate axis. Do not use these ICVs to configure the coordinate systems. Use them to query the coordinate system configuration so that motor motion can be specified in coordinate system units rather than encoder counts. Generally, these VIs will not be used at this level. Their capabilities are integrated into other collections of VIs.

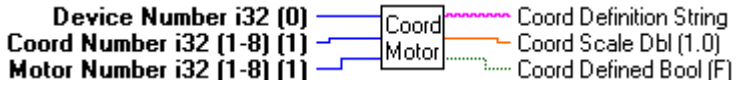
PmacTutor12 – Using Coordinate System Definitions

This exercise introduces ICVs for determining coordinate system definitions and parsing these definitions into LabVIEW data types that can be used to convert between encoder counts and coordinate system units and determining which motors, if any, are defined in a given coordinate system.

The architecture for the following PmacCoord ICVs is a little different from that used for status and other fundamental queries. A few of these VIs make use of local state variables to keep track of queries for coordinate system definitions so that these queries are not placed every time the VI is executed within your VI. This reduces communication traffic and relieves the developer from having to check for a new query. The fundamental assumption in this approach is that after the application has been created, it is not necessary to constantly redefine the specified coordinate system. When a coordinate system definition has been requested from PMAC and changed, the new definition will not be reflected in the application. To refresh the definition, temporarily request the definition for another coordinate number or close the VI and reopen it.

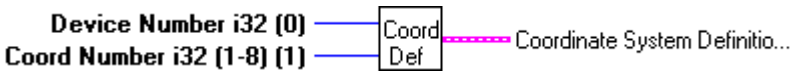
The architecture for determining coordinate system definitions relies on the following three VIs. They perform similar operations but return coordinate system definitions for different purposes.

- PmacCoordMotorDef – Query PMAC for the definition of Motor Number in Coord Number. If the motor is not defined in the specified CS Coord Definition is Encoder, Coord Scale = 1.0, and Coord Definition is false. If the motor is defined in the specified CS Coord Definition is true, Coord Scale is the encoder to CS unit scale factor, and Coord Definition is the definition (e.g. #1->16000X).



- PmacCoordScale – Query PMAC for the motors defined in Coord Number. The Coordinate System Scale Cluster (PmacCoordScale.ctl) contains three arrays with the motor definition, scale factor, and whether or not the motor is defined in Coord Number. The actual query is only placed if Coord Number changes from a previous call.

- PmacCoordDef – Fetch the motor scaling definitions for the specified coordinate system and provide a cluster for the PmacCoordDef indicator.



There are a few limitations to be aware of when querying coordinate system definitions from PMAC Panel Pro. Generally a motor is assigned to a single coordinate axis as in the following definition:

```
&1
#1->1000X
```

This specifies motor #1 as belonging to coordinate system &1 and that one X unit is 1000 encoder counts. The scale factor would thus be 1000.

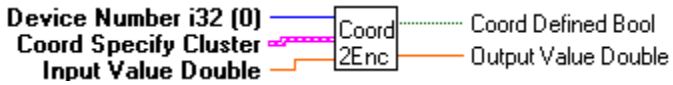
The limitation arises when coordinate system axes are linear combinations of several motors as in this example that rotates the coordinate system 30 degrees from the mechanical axes of the motor

```
&1
#1->8660.25X-5000Y
#2->5000X+8660.25Y
```

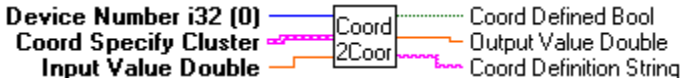
There is no easy way to parse this information when returned by PMAC and present it for use because there are so many possible ways of defining coupled motor systems. Furthermore, the individual items in the PmacMotors indicator clusters as defined would change the definition every time it is switched from encoder counts to CS units. If the axes are coupled like this, study the VIs presented here and modify them for custom use or hard code the scaling and motor state conversions into the application.

There are three VIs that use the scale factors provided by the previous VIs to convert numerical inputs from encoder counts to coordinate units and back. This can be done for individual motors or all motors in a coordinate system.

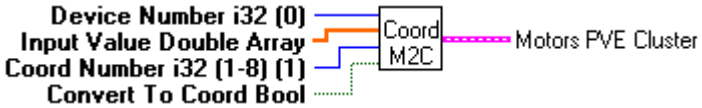
- PmacCoordMotorToEncoder – Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from CS units to encoder counts. If the motor is not defined in the CS no conversions is applied. If the motor is defined and Convert is true, Coord Defined is true, and Output Value is scaled from CS units to encoder counts.



- PmacCoordMotorToCoord – Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from encoder counts to CS units. If the motor is not defined in the CS, no conversion is applied. If the motor is defined and Convert is true, Coord Defined is true, and Output Value is scaled from encoder counts to CS units. Coord Definition is a string specifying Output Value units as Encoder or the CS definition of the motor.

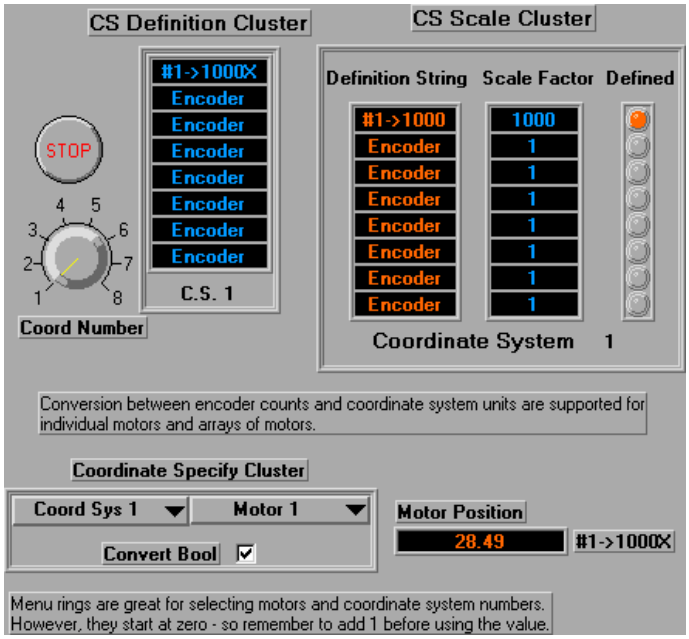


- PmacCoordMotorsToCoord – Generate an indicator cluster for PmacMotorsPVE.ctl. Input Value Double is an array of positions, velocities, or following errors from VIs in the PmacMotors collection. If Convert to Coord is true, fetch the CS definitions for the motors specified in Coord Number and scale them to CS units. Motors not defined in Coord Number are not scaled.



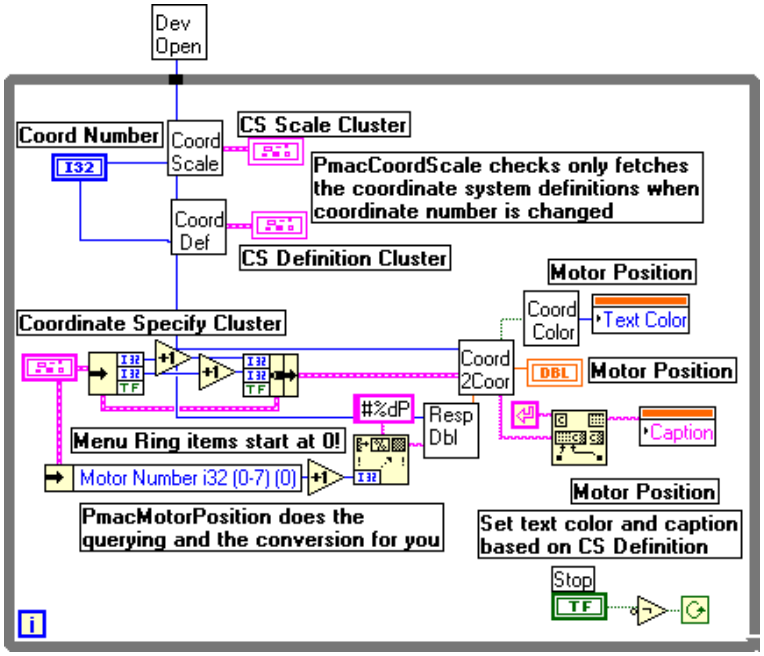
The panel shows two indicator clusters. The CS Scale Cluster contains the definition of all motors in the specified coordinate system as a displayable string, a numeric scale factor, and a Boolean indicating whether that motor is defined in the coordinate system. In the example, motor #1 is defined in coordinate system 1. The orange text color indicates that the CS defined in CS Scale Cluster's caption is being addressed. The CS Definition Cluster is a derivative of the larger cluster and can be used in conjunction with PmacMotorsPVE on your application panels.

The lower portion of the panel is a modified PmacCoordSpecify cluster used to specify a motor, coordinate system, and conversion from encoder counts to coordinate system units. The modifications were made by replacing individual control items in the stock cluster with types that are more appropriate. The orange numeric position indicator and its caption indicate the motor definition within the specified Coord Number.



The diagram for the query of coordinate system definitions is simple. The lower portion of the diagram demonstrates how to use the conversion VIs to convert and display the motor position data. The Coordinate Specify Cluster in this example is made from Menu Rings whose index always starts at zero. Because PMAC motors and coordinate systems start their number at one you must add one to the selection index. This is not necessary if using numeric controls in the Coordinate Specify Cluster.

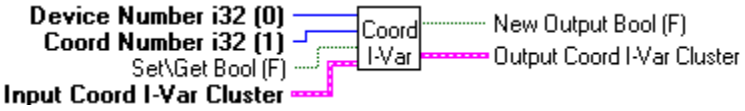
The motor position is processed by PmacCoordMotorToCoord to produce three outputs that can be used to enhance the display of the data. PmacCoordColor sets the color of the numeric indicators. The Coordinate Definition String is used to set the indicator's caption after stripping the terminating **<CR>**. The example shows the use of both named and unnamed unbundles to get the data required for the operation.



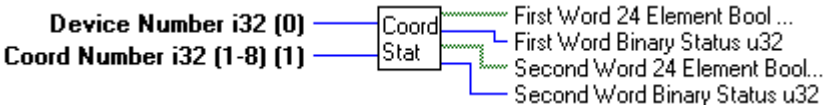
PmacTutor13 – Configuring and Monitoring Coordinate Systems

This exercise introduces ICVs for monitoring and configuring coordinate systems and program execution within coordinate systems. These VIs follow the same I-Variable and status architectures already introduced. The VIs are:

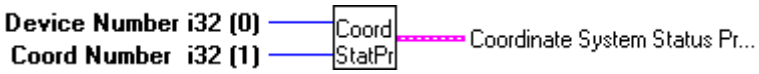
- PmacCoordIVar – Follow PMAC Panel Pro's standard I-Variable VI architecture. When Set/Get is true, the I-Variables for the specified Coord Number are set. Otherwise they are fetched from PMAC and provided by Output Coord I-Var Cluster with New Output true. Set/Get is not required and defaults to a Get operation.



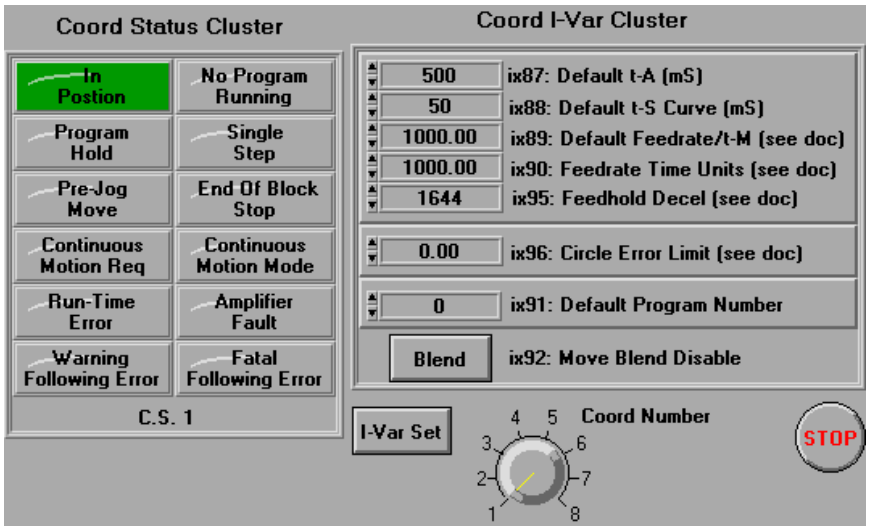
- PmacCoordStat – Query PMAC for the status of the CS specified by Coord Number. Report the two status words as arrays of Booleans and unsigned 32 bit integers.


Device Number i32 (0)
Coord Number i32 (1-8) (1)
Coord
Stat
First Word 24 Element Bool ...
First Word Binary Status u32
Second Word 24 Element Bool...
Second Word Binary Status u32

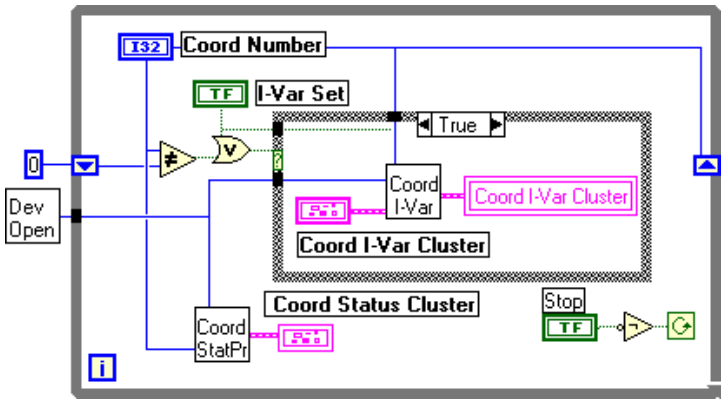
- PmacCoordStatProg – Create a status indicator cluster for the PmacCoordStatProg indicator containing the status for Coord Number.



Monitoring the coordinate system status is a very common operation because programs run in coordinate systems. If the coordinate system has Cartesian X-Y-Z axes and rotary U-V-W axes, then the program executes its moves using the motors assigned to those axes. The coordinate system status reflects the state of the executing program and the combined state of all motors in the system. If all motors are in-position then the coordinate system is in position. If any motor has a warning following error then the coordinate system has a warning error.



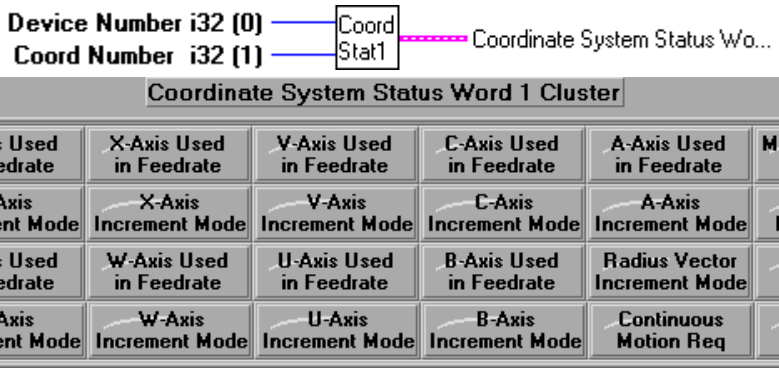
The diagram also has a familiar architecture.



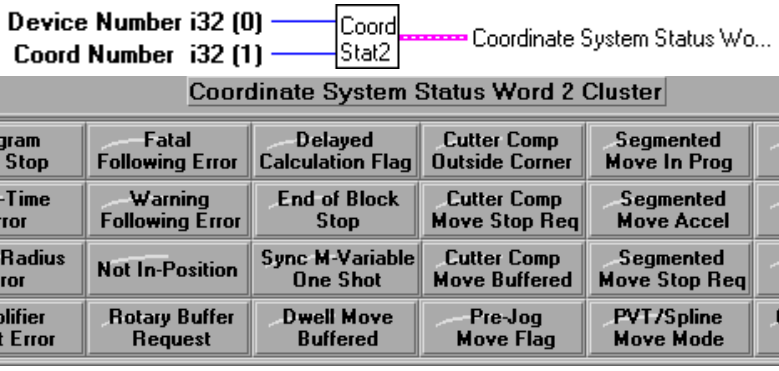
Coordinate System Status ICVs

There are, as expected, indicators for both coordinate system status words and a few miscellaneous VIs that will be introduced in later exercises.

- PmacCoordStat1 – Create a status indicator cluster for the PmacCoordStat1 indicator containing the status for Coord Number.



- PmacCoordStat2 – Create a status indicator cluster for the PmacCoordStat2 indicator containing the status for Coord Number.



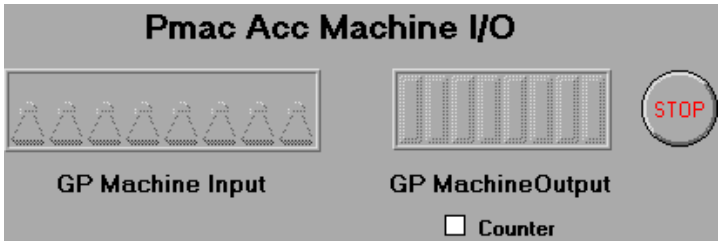
PmacAcc ICVs

This series of exercises introduce the PmacAcc collection of ICVs. These form templates that combine PmacMemory's direct access of memory with PmacIVar's Set/Get architecture that access PMAC's memory mapped devices. Using this approach, the bit field and address specifications can be hidden in the VIs.

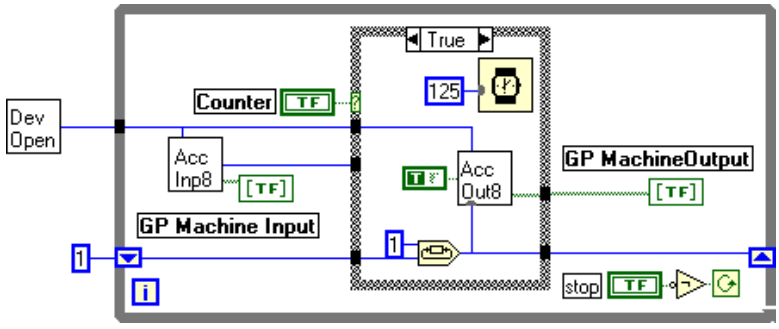
This collection of VIs will grow as Delta Tau adds support for its numerous accessories. The Machine Input/Output VIs and a simple example of the Acc-16D control panel will be demonstrated.

PmacTutor14 – Machine Input and Output

This tutorial demonstrates two VIs. The first one allows access to the general-purpose machine input port and one that accesses the output port. On the demonstration box used for the development of PMAC Panel Pro, switches drive the inputs and the outputs drive a set of LEDs. When running the example, the GP Machine Input indicator directly reflects the operation of the physical switches.

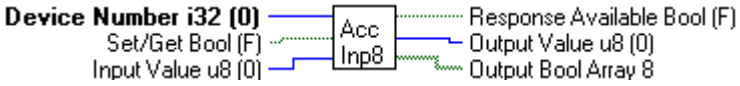


The physical inputs are copied into the output port located in a different bit field to drive the GP Machine Output array and the LEDs on the physical device. If the Counter box is checked, a bit is rotated through a number and written the physical port, thereby driving the LEDs. This is shown on the diagram below.

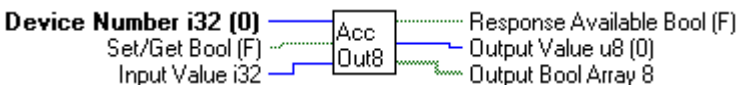


The two VIs that implement the example require a little discussion so that custom PmacAcc VIs can be developed or modified.

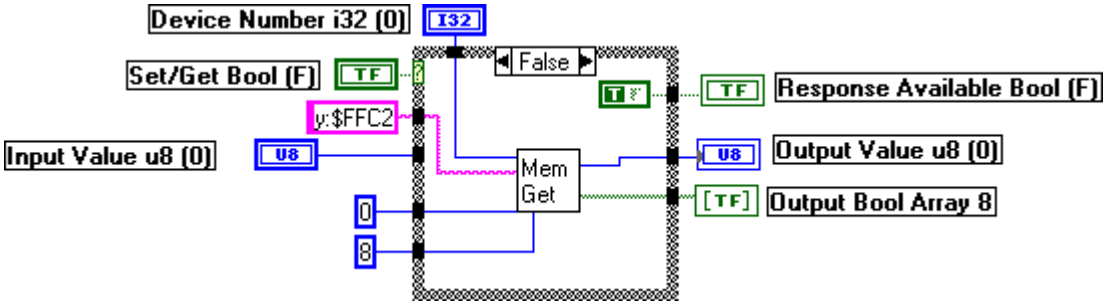
- PmacAccMachineInput8 – If Set\Get is false or not wired, get the Machine Input Port contents. Response Available will be true to indicate that the Outputs contain the value. If Set\Get is true, set the Machine Input Port using Input Value. Response Available will be false and the outputs default to Input Value.



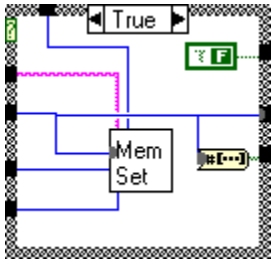
- PmacAccMachineOutput8 – If Set\Get is false or not wired, get the Machine Output Port contents. Response Available will be true to indicate that the Outputs contain the value. If Set\Get is true, set the Machine Output Port using Input Value. Response Available will be false and the outputs default to Input Value.



The diagram for PmacAccMachineInput8 is shown here to make a point about the PmacMemory VIs. The VI fetches the machine input port data located at Y:$FFC2 bits 0-8 using PmacMemoryGet.

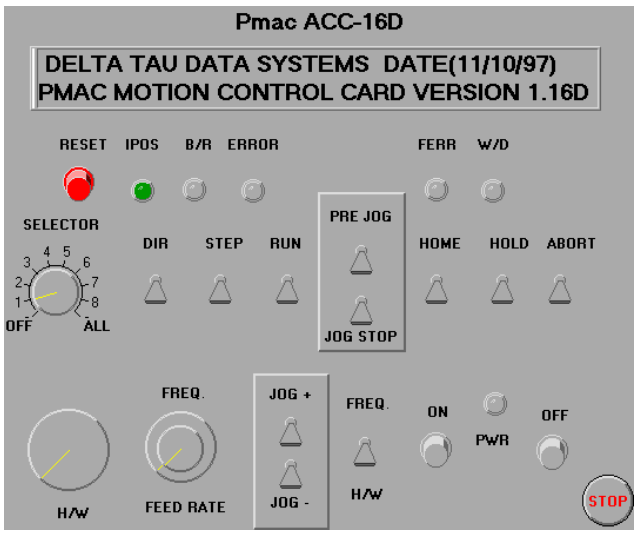


This case illustrates an important behavior associated with PmacMemorySet. This VI sets the contents of the bit field and provides as its output the entire 24-bit word. This was done so that multiple copies of the VI can be chained together to handle multiple bit fields. If the output of PmacMemorySet is taken and wired to Output Value for PmacAccMachineInput8, Output Value will not be what is expected. This is obvious when running PmacAccMachineOutput8. To remedy this, use the Input Value that is used to set the field as Output Value.

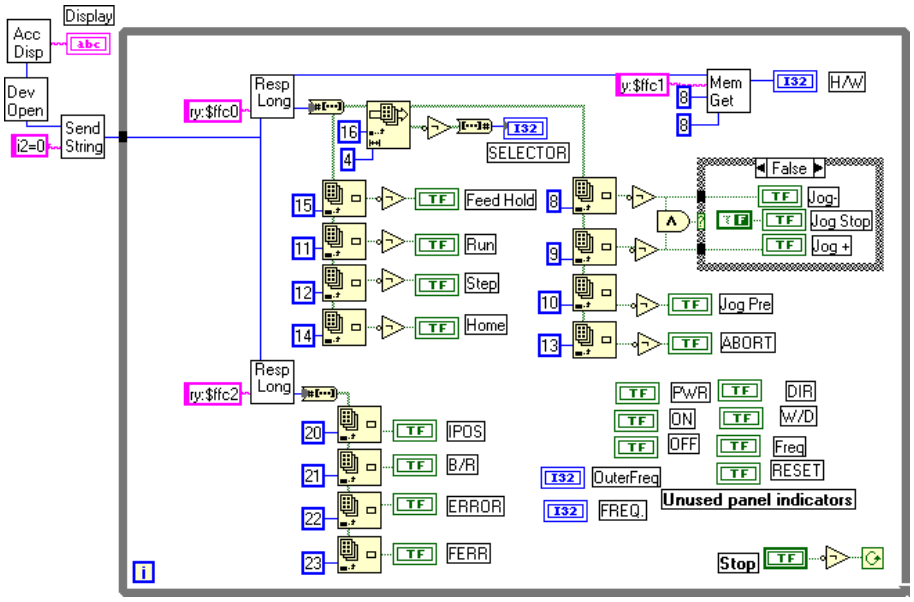


PmacTutor15 – ACC16D Control Panel

This example fetches the contents of the several memory locations for the Control Panel port at Y:$FFC0, Y:$FFC1, and Y:$FFC2. These registers allow the Acc-16D accessory to control the operation of motors and programs from an operator control panel. The panel for the example responds to the physical panel by mimicking the switch operation. If using an Acc-16D while running this example, the various switches on this panel will respond in kind.



The diagram shows that PMAC is queried for the contents of the three locations and then converted into appropriate types for processing. The selector field is extracted, as are the individual control/status bits. To make this panel control PMAC, use the same control layout and generate the appropriate commands by borrowing portions of PmacMotorJog, etc.



DEVELOPMENT TOOLS

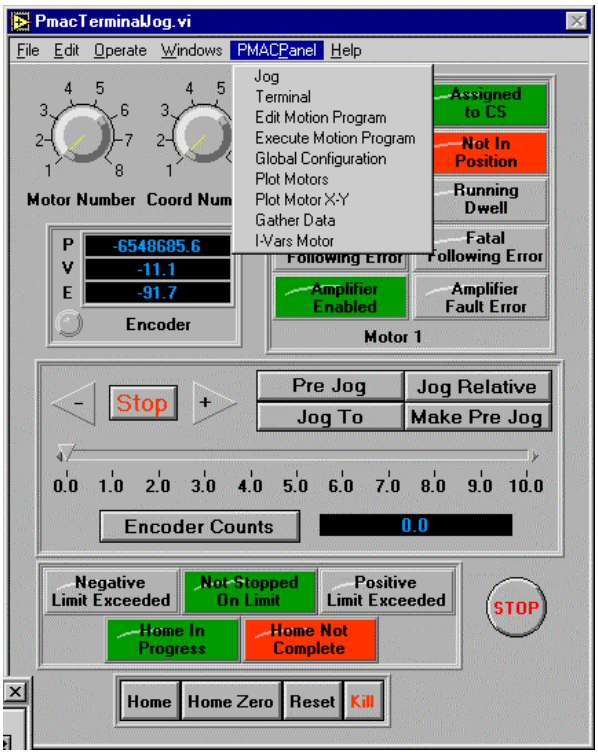
Serious PMAC configuration, tuning, and setup require the use of Pewin 32 Pro. Once this step is completed, development of the PMAC Panel Pro application can begin. PMAC Panel Pro supplies a number of tools and application VIs to aid in this process and provides architecture for adding more.

There are ten standalone development tools covered in this chapter. They make extensive use of the ICVs introduced previously and form an excellent introduction to the PMAC Panel Pro integration ICVs introduced later in this manual.

- **PmacTerminal** – A basic ASCII terminal with useful ICVs for monitoring the coordinate system and motor status. In addition, several new ICVs for controlling programs and PLCs are introduced.
- **PmacTerminalEdit** – A simple editor for program development and downloading. The tool also supports the creation of encapsulated motion and PLC program VI wrappers that hide the details involved in controlling and monitoring PMAC motion and PLC programs.
- **PmacTerminalExecute** – An interactive debugger for monitoring and controlling the execution of PMAC programs.
- **PmacTerminalJog** – A simple tool for jogging and controlling motors
- **PmacTerminalMotors** – A graphical tool for monitoring and plotting the motion of multiple motors in a real-time strip chart.
- **PmacTerminalMotorsX-Y** – A tool for monitoring and plotting the motion of motors in a real-time X-Y chart.
- **PmacTerminalGather** – A tool for specifying and gathering PMAC motion data and exporting it to Microsoft Excel.
- **PmacTerminalMotorIVars** – A tool for configuring individual motor I-Variables.
- **PmacTerminalCoordIVars** – A tool for configuring coordinate system I-Variables and monitoring coordinate systems.
- **PmacTerminalGlobal** – A tool for monitoring PMAC's state, saving configurations, and configuring important global I-Variables.

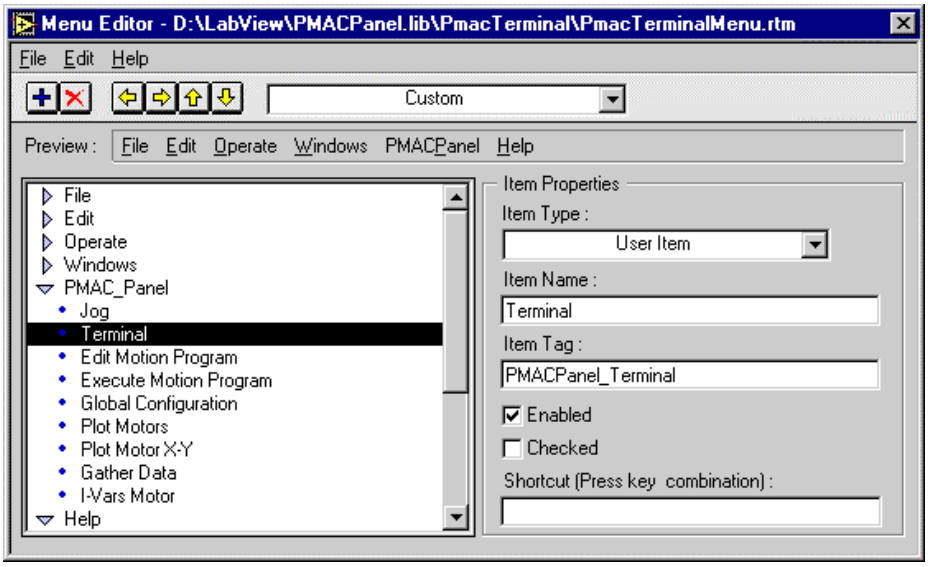
Tool Menus

The tools distributed with PMAC Panel Pro can be integrated into an application using LabVIEW 5.0's new ability to edit and process menu bar selections. This allows the applications easy access to the development tools and an easy way to add custom tools. The figure below shows the panel for the PmacTerminalJog tool. The menu bar for the tool shows an entry for PMAC Panel Pro that contains the menu items for the development tools.



Modifying the Menu

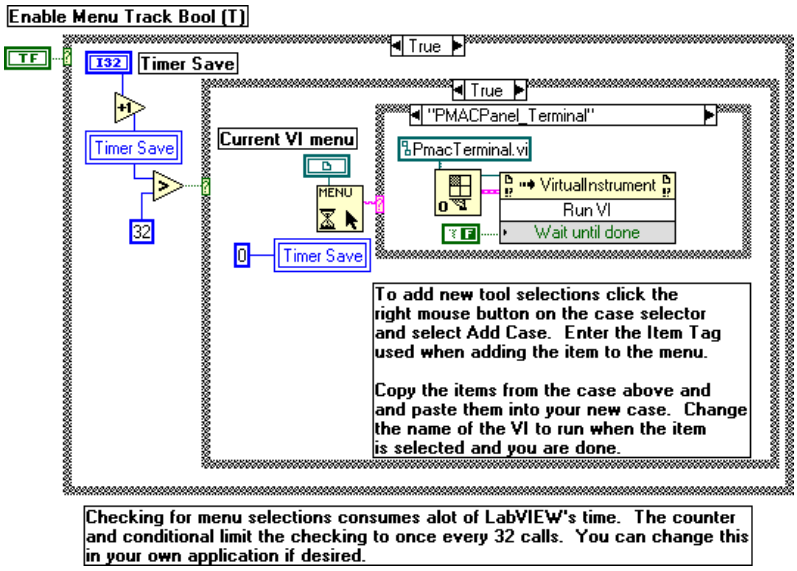
This is a custom menu named PmacTerminalMenu.rtm that is set as the default for the application VIs run-time menu. To modify the application VI's menu or modify the existing one use the **Edit»Edit Menu** option from the VIs panel to display the following dialog.



Select **File»Open** to display the selection box, then double click on **\PmacTerminal\PmacTerminal Menu.rtm** or click **OK**. This will change the VI's run-time menu from the default or minimal options to custom. Note that the panel shows an Item Name as it appears in the menu and an Item Tag that is used by PmacTerminalMenu.vi to decode the selection and execute the desired tool. To add custom tools to the menu, highlight an existing menu item entry and select **Edit»Insert User Item**. Give the new entry a name and a tag. Save the modifications. All existing PMAC Panel Pro tools using the menu capabilities will now reflect the changes. If a new tool has been added, some additions must be made for the new VI.

Modifying PmacTerminalMenu

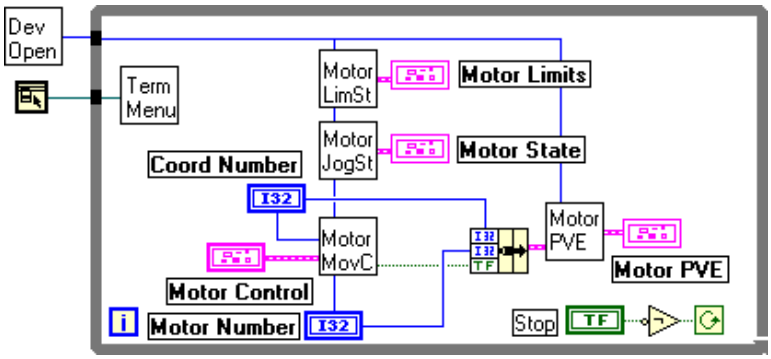
To actually implement the new selection, add it to PmacTerminalMenu. The diagram for the VI is shown here. The instructions indicate to add a case for the Item Tag specified in the Edit Menu panel and to copy the VIs to execute the VI into the new case. Once completed, all tools can access each other without needing any new configuration.


Enable Menu Track Bool (T)
TF
I32
Timer Save
True
True
"PMACPanel_Terminal"
Current VI menu
PmacTerminal.vi
MENU
VirtualInstrument
Run VI
Wait until done
Timer Save
+1
32
0
To add new tool selections click the right mouse button on the case selector and select Add Case. Enter the Item Tag used when adding the item to the menu.
Copy the items from the case above and and paste them into your new case. Change the name of the VI to run when the item is selected and you are done.
Checking for menu selections consumes alot of LabVIEW's time. The counter and conditional limit the checking to once every 32 calls. You can change this in your own application if desired.

There are a few limitations associated with using LabVIEW's menubar capabilities. The main one is that it slows applications noticeably that are doing highspeed query/response updates of status and position indicators. To remedy this, the PmacTerminalMenu VI is configured to execute the actual checking of the menu selection once every 32 calls. This logic can be changed or add timers to suit individual needs.

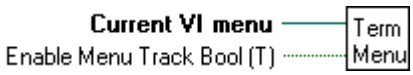
Basic Tool VI Requirements

PMAC Panel Pro's use of the menu bar selection VIs spawns the VI specified by the selection as a separate concurrently executing task. This means the VI should be organized much like the exercises. Open PMAC, run a while loop, use the **Stop** button. The diagram for PmacTerminalJog is shown here to illustrate this.



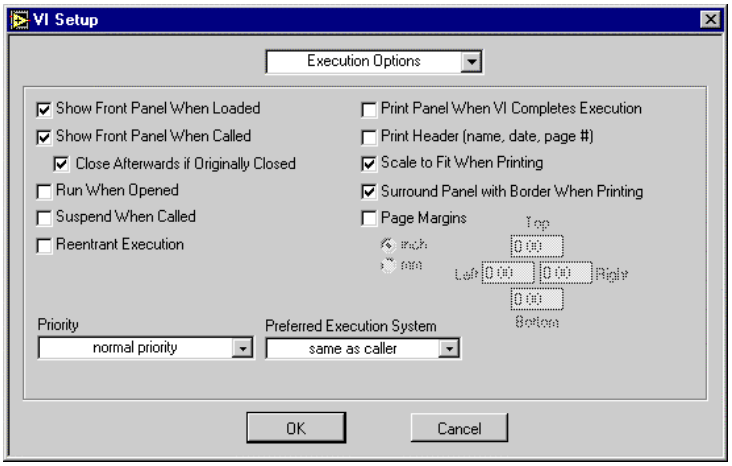
The only modification is the addition of the PmacTerminalMenu VI that actually spawns the new VI in response to a menu selection and the Current VIs Menu function selected for the function palette.

- PmacTerminalMenu – This VI uses the Current VI Menu supplied to it to check for selections. Because there are time critical operations that should not check for selections, set Enable Menu Track false using a switch to disable the testing. This is done in the PmacTerminalMotors tool because the delay causes a noticeable blip in plotted position data.

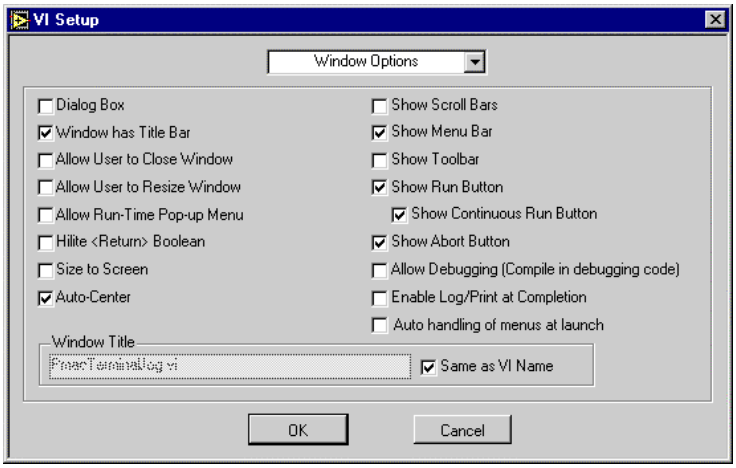


Basic Tool VI Configuration

When a tool is spawned, the Execution and Window options defined for the VI Setup are important. The following panel shows the Execution Options used for a PMAC Panel Pro tool. The panel should be shown when loaded, otherwise selecting the tool runs it without a panel. If the VI is called instead of spawned then use the Show Front Panel When Called and then close it afterward. The problem with calling the VI rather than spawning it is that it must complete before the caller's panel becomes active again.

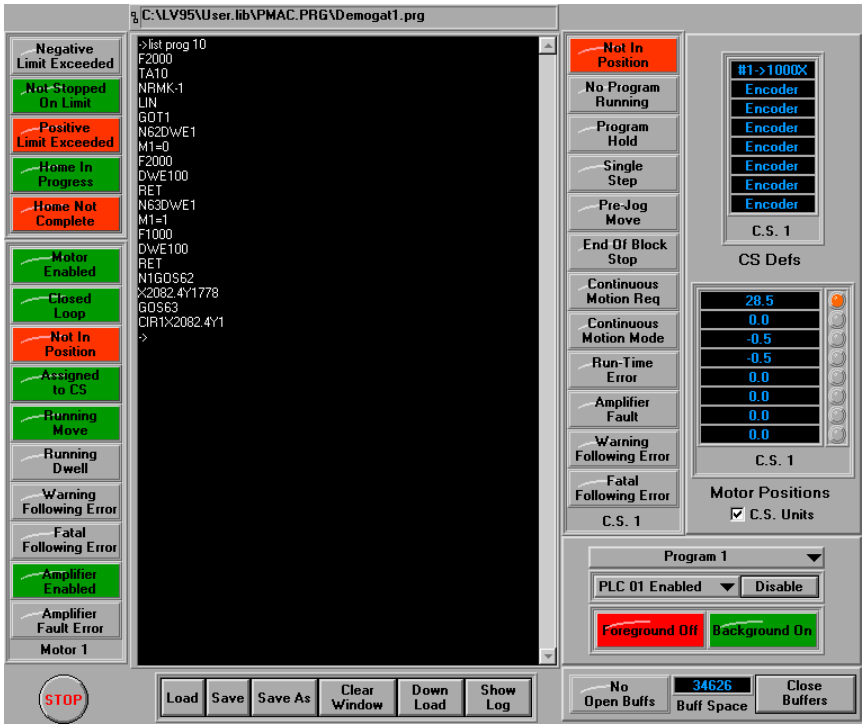


Generally, the panel window should prevent users from doing too many things. PMAC Panel Pro tools have a title bar and auto-center. When in edit mode, the menu bar, run button, and abort button are shown. When running, these disappear and the menubar of the window just installed must be used.



PmacTerminal

PmacTerminal is a command-line terminal tool like Pewin 32 Pro. The panel says a lot about its operation. As the tool is explained, many tricks can be used to build better PMAC Panel Pro applications.



Basic Terminal 101

The black screen is a multi-line string with a scrollbar configured for normal operation. Typed commands are sent to PMAC when the **<RET>** key is pressed. LabVIEW does not do this automatically. To accomplish this, there is a Boolean button named **OK** hidden behind the string control that has its Key Navigation set to capture the **<RET>** key. Select **Project»Find** and locate OK and it will display. Setting the Key Navigation this way means that when **<RET>** is pressed anytime the cursor is in the window, the **OK** button is activated. There is a lot of spaghetti diagram to keep track of the current line, character position, and other book keeping that fetches the line just entered from the multi-line string and sends it to PMAC using PmacCommRespStr. The terminal indicates it is expecting a command with a -> prompt on the screen.

This process puts the Key Focus on the **OK** button, thereby removing the cursor from the screen indicator. Focus is returned to the screen indicator by creating an attribute node for the screen indicator, selecting Key Focus, and setting it true every time the OK case is executed.

Another issue that arises in a terminal-like string control is LabVIEW's use of **<TAB>** to give Key Focus to other panel controls according to the Panel Order. This cannot be disabled so understand that hitting **<TAB>** throws the cursor out of the screen and onto the **OK** button, then the **Stop** button, etc. as defined by the Panel Order. Control character sequences work but do not display as expected. If **<Ctrl + A>** is typed, only an **a** appears on the screen. Hitting return does indeed send the **^a** command to PMAC and all program motion will abort as expected.

Basic Command Editing

Use the standard cut, copy, and paste control sequences to manipulate text in the screen buffer. Copy a previous command, paste it at the end of the buffer, and execute the command with a **<RET>**. In addition, copy PMAC responses to other applications or other LabVIEW string controls. Text cannot be inserted into the middle of previously executed commands. The spaghetti diagram does not know where the insert took place and will require a lot of work to track this type of operation.

If entering lines that wrap to the next line on the screen, list the programs with lines that wrap, or list the gather buffer (guaranteed to wrap), the screen may start to act a little strange. Solve this by clearing the window using the button below the screen.

Buffer Management

PMAC Panel Pro applications, especially their development, require the management of ASCII buffers containing command sequences, I-Variable configuration information, and motion/PLC programs. To simplify this, PMAC Panel Pro has a VI and associated cluster control to handle ASCII text buffers. The control is located below the screen. It is a copy of the \PmacProgram\PmacProgEdit control shown below. The actual control on the PmacTerminal panel does not look like the raw control. Be careful when moving items in a cluster. Do not pull them out of the cluster boundaries by mistake. This causes the remaining controls in the cluster to reorder their Panel Order. The associated VI will not work as expected because what was control 4 is now control 5, etc.

The cluster and VI implement six operations using the screen buffer:

- **Load** – Load an ASCII file into the screen buffer.
- **Save** – Save the contents of the screen buffer to the file specified in the path control above the screen.
- **Save As** – Save the buffer by selecting a new file.
- **Clear Window** – Dump the buffer and reset the display bookkeeping.
- **Down Load** – Save the screen to a temporary file, compile the file, and down load it to PMAC. This means that the entire buffer must contain legitimate on-line commands and/or a program. Buffers with previous PMAC responses are not downloadable.
- **Show Log** – When compiling the buffer PComm 32 Pro generates a log file with standard compiler messages. If the download generates an error, a dialog with the log file is displayed. Use this button to review the log file whether or not an error was generated.

Terminal Indicators

There are several indicators introduced in previous sections on the panel that display the status of the motor, coordinate system, and motor motions. These are simple rearrangements of the stock clusters. The interesting thing about these clusters is that they track the commands entered on the terminal. Address Motor 3 by typing **#3** on the terminal screen and the motor clusters display the status for motor 3. Address coordinate system 2 by typing **&2** on the terminal screen and the coordinate system status clusters display the status for coordinate system 2.

Terminal Controls

The PmacGlobalBufferSize VI is used to monitor and control PMAC's buffer status. Sometimes when using a terminal program its not obvious that a download cannot succeed because a buffer was left open or the gather buffer was not deleted. The size indicator and the Close Buffer button make it easier to monitor and deal with this problem.

There are three new control clusters and capabilities used to develop this tool.

Operationally these VIs are:

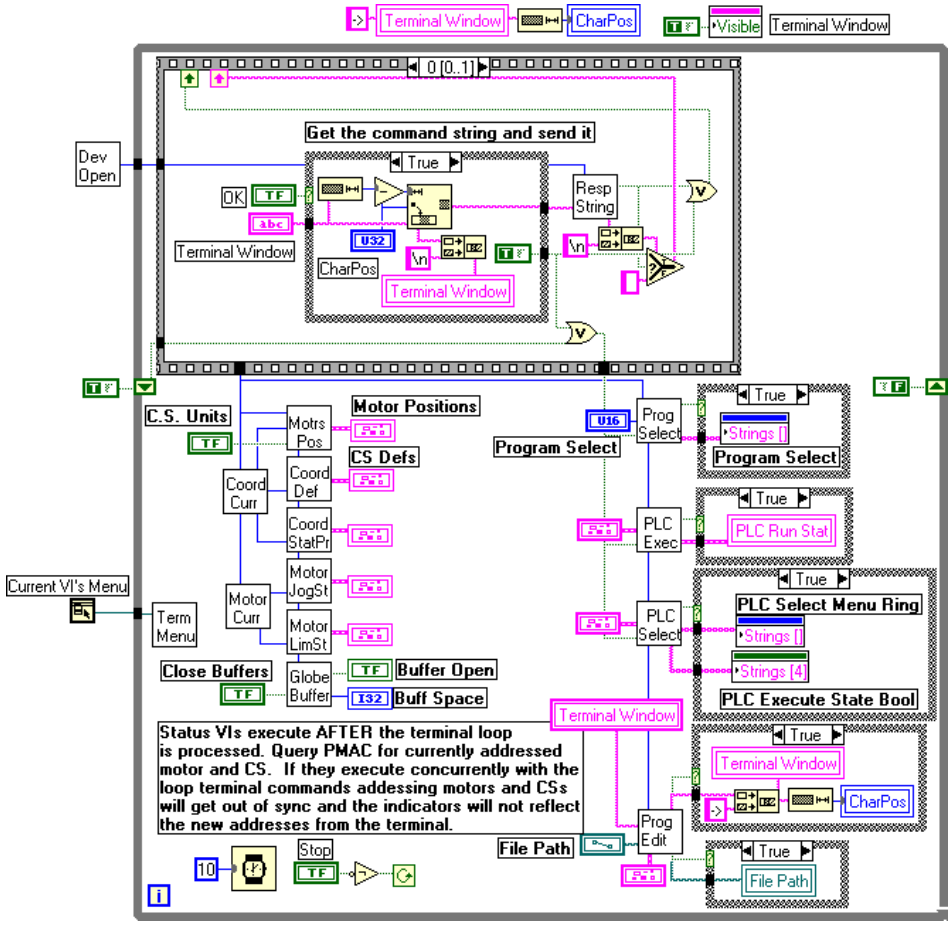
- PmacProgram\PmacProgSelect – This menu ring control with its VI query PMAC for a list of currently loaded motion programs and display the data in the ring. Using it, it can be determined what programs are currently loaded in PMAC.
- PmacPLC\PmacPLCSelect – This cluster contains a menu ring control that works with its VI to query PMAC for a list of currently loaded PLC programs and their execution status: Enabled/Disabled.

In addition, a button toggles the Execution State of the selected PLC. If the menu ring shows PLC 01 as Enabled, use the button to disable it thereby changing the text displayed in the menu ring.

- PmacPLC\PmacPLCExecute – I5 controls the general execution of PMAC PLCs. Bit 0 controls the execution state of PLC 0 (the foreground PLC) and Bit 1 controls the execution state of PLCs 1 - 31 (the background PLCs). This control and its VI maintain I5. When the button says Background off, the background PLCs are off. Clicking on the button toggles them on and it will now indicate Background On.

Implementation Diagram

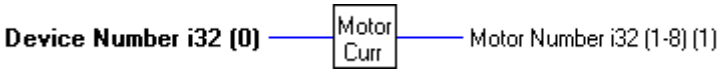
The diagram for this tool illustrates some important concepts in the development of tools and custom applications.



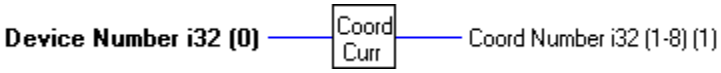
The general framework should look familiar. There is an execution loop, a PmacDevOpen, a Stop button, and the PmacTerminalMenu items. Several bookkeeping local variables are initialized outside the loop.

Most of the status indicator VIs are located in the lower left. The currently addressed motor and coordinate system are fetched by the VIs

- PmacMotorCurrent – Query PMAC for the currently addressed motor. Generally it is used in interactive development environments rather than a custom VI to address a specific motor.

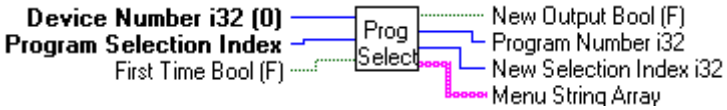


- PmacCoordCurrent – Query PMAC for the currently addressed coordinate system. Generally, it is used in interactive development environments rather than a custom VI to address a specific CS.



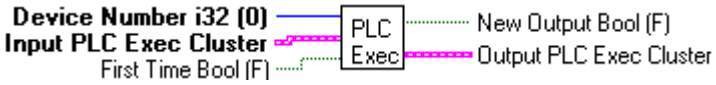
The VIs and spaghetti diagrams on the lower right implement the PmacProgEdit, PmacProgSelect, PmacPLCSelect, and PmacPLCExecute operations. These utilize the update architecture used in many of the earlier exercises. The VIs accept control clusters containing Booleans and generate new output data for the controls when an output Boolean indicates it has new data. Several of the clusters function as both controls and indicators using their color and Boolean text. This manual does not cover the internal operation or implementation of these VIs or members of these collections in detail. They are involved and way beyond what most developers will want to know about PMAC Panel Pro. Feel free to examine their contents and make changes as desired.

- PmacProgSelect – Query PMAC for a description of all loaded motion programs by reading PMAC's internal buffers. If First Time is true, Menu String Array contains a sorted list of all loaded programs by program number. The VI maintains New Selection Index as a state from execution to execution. Translation of Program Selection Index into Program Number occurs when First Time Strings is true or Program Selection Index is not equal to New Selection Index. New Output true indicates that Program Number, New Selection Index, and Menu String Array contain new data.

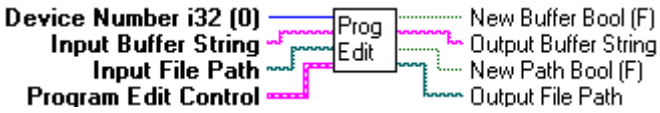


- PmacPLCSelect – Query PMAC for a description of all loaded PLC programs by reading PMAC's internal buffers. If First Time is true, Menu String Array contains a sorted list of all loaded PLC programs and their execution state by PLC number for the menu ring in PLC Select Cluster. Button String Array contains information to change the description of the button in PLC Select Cluster so that it toggles the selected PLC's execution state when clicked. The VI maintains New Selection Index as a state from execution to execution. Translation of menu ring selections in PLC Select Cluster into PLC Selected Cluster occurs when First Time Strings is true or either control in PLC Select Cluster changes. New Output true indicates that PLC Selected Cluster, Menu String Array, and Button String Array contain new data.

- PmacPLCExec – This VI controls the execution of foreground and background PLC programs by modifying i5 using a PmacPLCExec control as both an indicator and a control. When First Time is true, New Output is true and Output PLC Exec Cluster indicates the state of foreground and background PLC program execution. When the buttons in Input PLC Exec Cluster do not match, the last Output PLC Exec Cluster contains the execution state of the foreground or the background PLC programs are toggled.



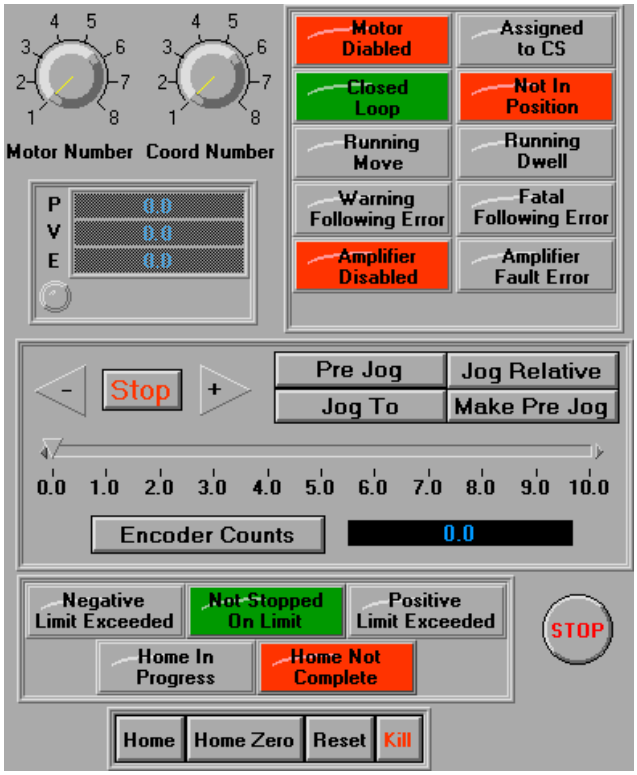
- PmacProgEdit - Manage common editing operations on Input Buffer String as specified by Program Edit Cluster. Input File Path is the default file path to use for Load, Save, or Save As operations. New Output Buffer is true when a Load or Clear Window operation puts new data in Output Buffer. New Path is true when a Load, Save, or Save as operation modifies the file path.
  - **Load** – Load a file into Output Buffer.
  - **Save** – Save input buffer to Input File Path.
  - **Save As** -– Query the user for a new file to save Input Buffer.
  - **Clear Window** – Put an empty string in Output Buffer.
  - **Down Load** – Compile and down load Input Buffer to PMAC.
  - **Show Log** – Display the contents of the compile log.



Using these descriptions, it is straightforward to use these VI's powerful capabilities. Place the appropriate control or controls on the panel. Where required create a writeable local variable copy of the control or the required items attribute node. If the menu ring requires an attribute node to display the programs, or the button requires an attribute node to change its Boolean text, create the node, and select the proper attribute. For the PLC Select cluster, go to the panel and create the attribute node for each item in the cluster – not the cluster itself.

PmacTerminalJog

This tool is a modified version of PmacMotorMoveExamp and PmacTutor7. Its details will not be covered here. Instead, it is an excellent example to demonstrate the behavior of multiple executing tools and application VIs. One note of importance is the limits on the numeric slider in the PmacMotorJogControl cluster. Change the limits to reflect the limits of the mechanical setup.



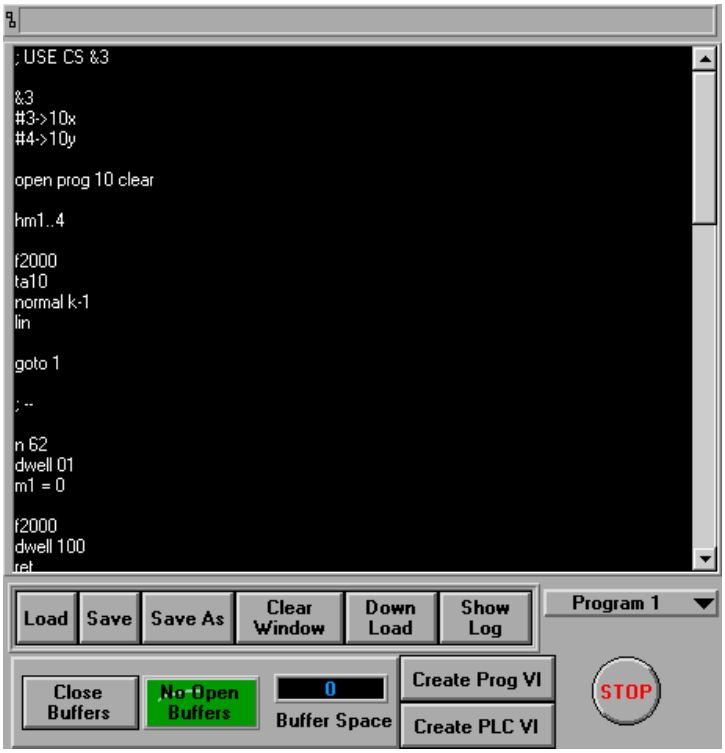
Start by opening and executing PmacTerminal. As motors and coordinate systems are addressed, the captions in the terminal tool indicators change. If PmacTerminalJog is executed by opening and running it or by selecting **PMAC Panel Pro»Jog** from the VI menu, things start to behave strangely. If the last few PmacTerminal commands addressed motor #3 and coordinate system &2, everything looks fine on the indicator bars.

When PmacTerminalJog is run, the indicator bars suddenly reflect the status of motor #1 and coordinate system &1. A terminal command was not sent to do this. The truth is that there was a command sent. There is only one PMAC and any commands that require a motor or coordinate system address change the addressed item. Because PmacTerminal queries PMAC for the currently addressed motor and coordinate system, it will use these values for its status queries. If these VIs are replaced with numeric controls, this will no longer be a problem for PmacTerminal. However, it will not be possible to have the indicators track the commands PmacTerminal sends to PMAC automatically.

PmacTerminalEdit

If all the fancy stuff is stripped from PmacTerminal and only edit control, program menu ring and a window, are left, it is now a program editor tool. The terminal-like interface is different in that there is no OK button to capture and process the line just entered. Instead, hitting **<RET>** in the screen puts a **<CR>** in the buffer. The **<TAB>** command will still rotate through the controls.

This tool should be used to develop programs rather than PmacTerminal. Program development is detailed in the PMAC User Manual and PMAC Software Reference Manual. There are some added features to PMAC Panel Pro's processing of motion and PLC programs developed using this tool. PMAC Panel Pro parses the motion program buffer for the PLC or motion program number and coordinate system prior to down loading. This allows the writing a PLC or motion program and having PMAC Panel Pro keep track of this information. It simplifies the PMAC Panel Pro diagrams. The program shown in the panel demonstrates what is required to do this.



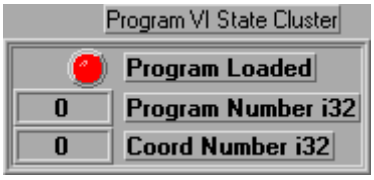
The motion program coordinate system is specified by the comment line:

```
; USE CS &3
```

The down load compilation process ignores comments so this does not affect normal PMAC operation. The comment can be any mix of upper and lower case. For safety, always use capital letters. The program number is parsed from the line:

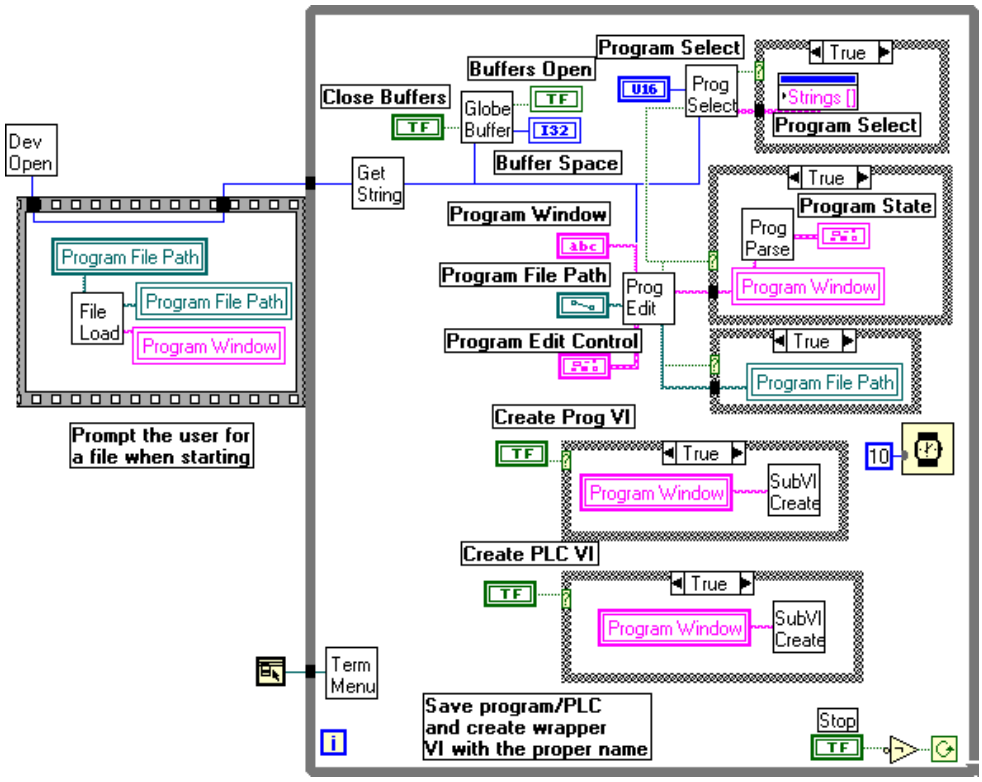
```
open prog 10
```

Again, this can be a mix of upper and lower case. When the program is down loaded, a cluster indicating the coordinate system to use, program number, and Load State is created:



This cluster is used by a number of PmacProgram VIs to do useful things.

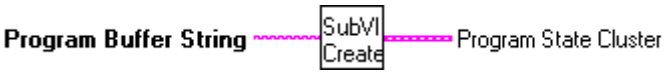
The diagram for the tool has a standard organization. Note the use of PmacProgParse to parse the program window and generate the Program State cluster. This cluster is hidden on the panel and not used in the tool.



Encapsulating Motion Programs

The Create Prog VI and Create PLC VI case structures at the bottom pass the program buffer to a very special VI that creates a VI wrapper for the program being edited. The detailed use of the new motion or PLC program VIs will be covered in later sections.

- PmacProgSubVICreate – Prompts for a file to save the Program Buffer to. Make a copy of PmacProgSubVI VI and change the name to the same base as the saved program. For example, if Program Buffer is saved to PmacTest1.pmc a new VI named PmacTest1.vi is created from PmacProgSubVI.



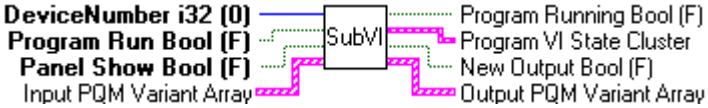
Using the example names above, there are now two items with the same base name. An ASCII program file PmacTest1.pmc and a VI PmacTest1.vi. The icon for the new VI based on PmacProgSubVI shown here.

- PmacProgSubVI – PmacProgSubVICreate makes a copy of this VI with a new name that matches the name of a motion program. Because the motion program has the same name (with a different extension), this VI knows how to open, parse, load, and run a motion program without intervention or extra inputs. It allows the editing of the associated program and interactively executes the program. Details of its implementation are contained in the manual.

The VI downloads the associated program when first loaded unless this option is disabled in the diagram and defaults for Program Number and Coord Number are provided for the Program VI State Cluster.

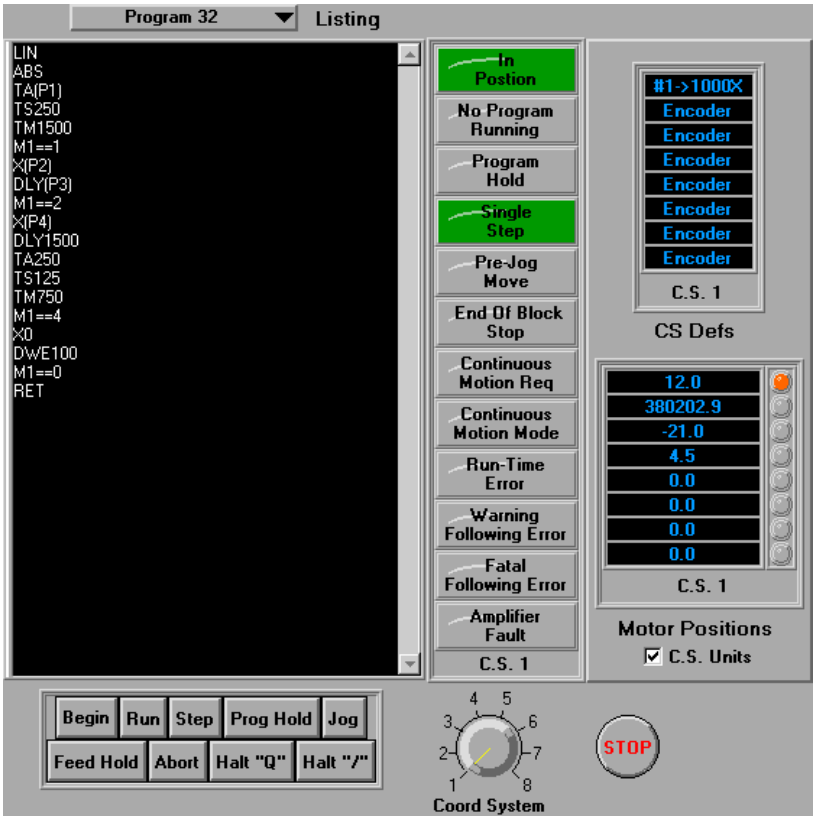
The interactive panel can be opened and used by setting Panel Show (latched) true. See the documentation for PmacTerminalEdit and PmacTerminalExecute for details on interactive execution. Click the **Stop** button on the panel to close it.

When the latched input Program Run is true, Input PQM Variant Array is sent to PMAC to initialize a program's P, Q, or M-Variables. The program is then started as long as there is no program executing in the associated CS. When Program Running is true, this or another program is executing in the associated CS.



PmacTerminalExecute

This tool is an interactive motion program debugger that queries PMAC for the actual program and allows stepping through it in a variety of ways. Querying PMAC for the program allows seeing whether the program is actually what it is. If a program is donloaded without first clearing the program buffer, the down load contents are appended to the existing buffer. The indicators and their operation should be obvious. Extensive on-line help is available for the panel controls using LabVIEW's **Help»Show Help** facility.

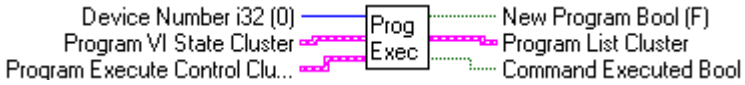


To operate the tool, select a program using the menu ring and a coordinate system to execute the program in using the knob. Only loaded programs are shown in the ring. Click **Begin** and the Program Execute cluster loads the program into the list buffer from PMAC. Click **Run** and the program will begin executing. Watch the execution by clicking the cursor in the screen. The currently executing program line will highlight and track the program through its steps. This dynamic display only works when the cursor is in the screen.

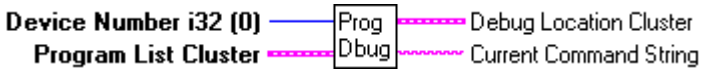
A few issues about PMAC program execution must be understood. PMAC pre-computes moves several lines ahead of the currently executing motion to allow motion blending. Because of this, the highlighted line may not reflect the moves the machine is currently executing. For detailed information on this, see the PMAC User Manual.

The implementation of the tool uses two VIs to generate program execution commands and monitor the currently executing program line. These VIs are,

- PmacProgExec – Interactively executes the program specified in Program VI State Cluster in response to button clicks in Program Execute Control Cluster. New Program is true when Begin is clicked and Program List Cluster contains a new listing. Command Executed is true when any button in Program Execute Control Cluster is clicked.



- PmacProgDebug – Query PMAC for currently executing line in Coord Number and output the response in Current Command. Determine the scroll position and characters that delimit this line in List Buffer and create Debug Location Cluster for setting selection attributes in a multi-line string control for real-time display of Program Number's execution. This information is obtained from PMAC using the **`LIST PE`** command.



The specification of the currently executing line in the program-listing buffer is given by:

Debug Location Cluster – Cluster of information for string control attributes. The items define the Scroll Position of the string in the buffer and the Start and End Character of the line currently executing.

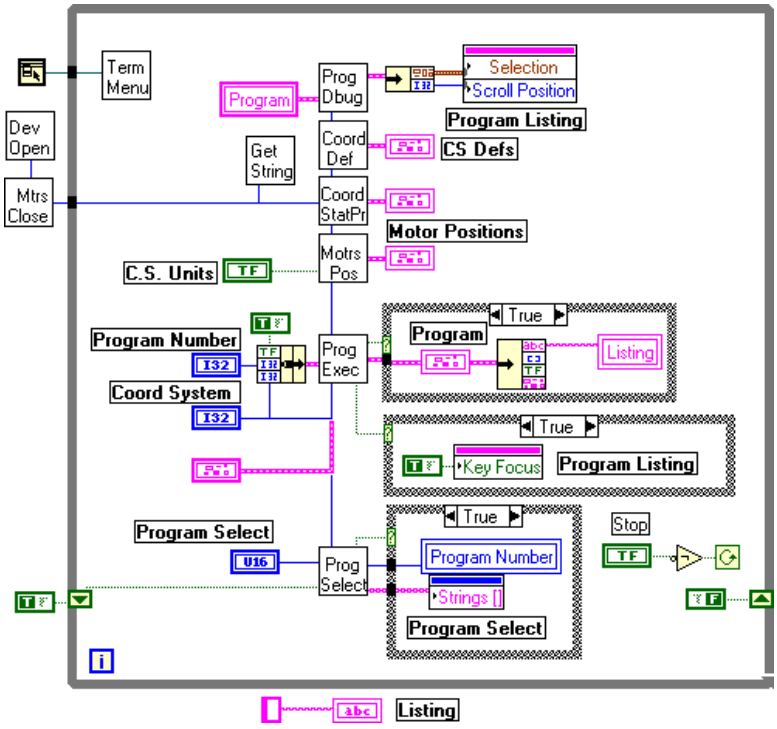
Selection Start and End character in List Buffer for currently executing program line.

Character Start

Character End

Scroll Position Number of the line executing currently in List Buffer.

The information in this cluster is used in the diagram to set the selection and scroll position attributes for the string control used to display the listing. This can be seen at the top of the diagram.



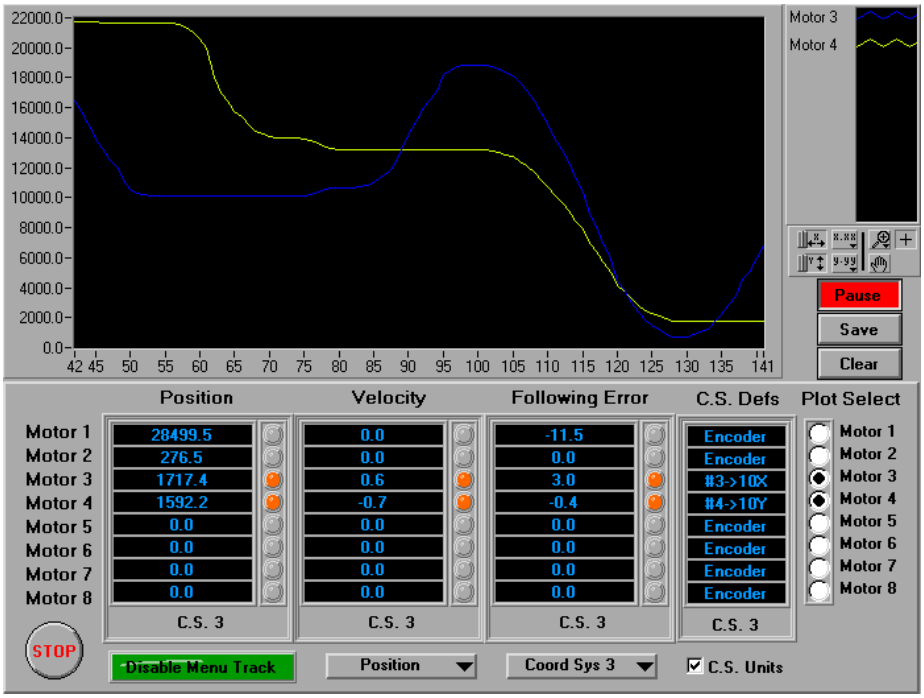
The program controls implemented by the Program Exec cluster send on-line program execution commands to PMAC. A brief description of the button operations is given here. For detailed descriptions of PMAC's implementation of the command, see the associated documentation in the PMAC Software Reference Manual or the online help available through **Help»Show Help**.

- **Begin** – Point PMAC to the coordinate system and program number specified. Load the actual program from PMAC. The command sent to PMAC is **&*CS*b*PROG*** where *CS* is the specified coordinate system number and ***PROG*** is the current program number.
- **Run** – Execute the program from where it is. If pointed to the beginning with Begin, then start there. If aborted or motion is halted using the associate buttons, restart the program from its current location. The command sent to PMAC is **R**.
- **Step** – Execute a step to the next move or dwell in the program performing all the intervening computations. The command sent to PMAC is **R**.
- **Prog Hold** – Bring the coordinate system velocity to zero, thereby holding moves where they are but allowing jogs. Restart the program with **RUN** or **STEP**. The command sent to PMAC is ****.
- **Jog** – Pop up a jog panel to allow motor jogging after a Prog Hold. The jog panel must be closed before the panel before returning to this panel to restart the program with **RUN** or **STEP**. Closing the panel returns all jogged motors to their pre-jog position with **j=**. Commands sent to PMAC are generated from the panel.
- **Feed Hold** – Bring the coordinate system velocity to zero thereby holding moves where they are. Jog cannot be done from this mode. Restart the program with **RUN** or **STEP**. The command sent to PMAC is **H**.
- **Abort** – Decelerate all motors immediately. Restart the program with **RUN** or **STEP**. The command sent to PMAC is **A**. See the safety notes for the command in the PMAC Software Reference Manual.
- **Halt Q** – Quit the program at the end of the current move or any already calculated moves. Restart the program with **RUN** or **STEP**. The command sent to PMAC is **Q**.
- **Halt /** – Quit the program at the end of the currently executing move. Restart the program with **RUN** or **STEP**. The command sent to PMAC is **/**.

PmacTerminalMotors

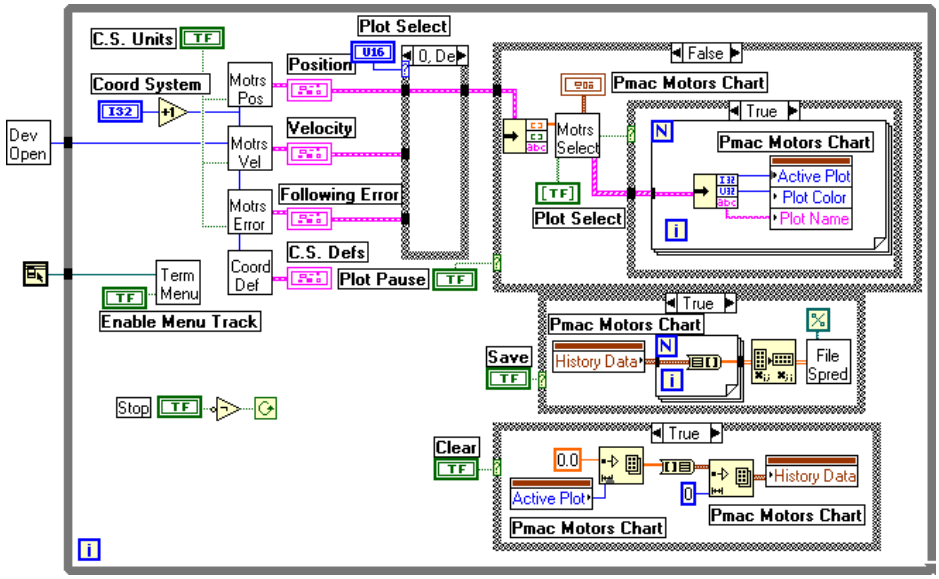
This tool is based on PmacTutor10 that introduced the PmacMotors ICVs. Its purpose is to monitor selected motor motions from the terminal. The arrays of indicators function as introduced in the earlier tutorial. Select a coordinate system and enable CS conversion for motors defined in the specified coordinate system. Motors selected in Plot Select are displayed in the real-time chart. The sampling rate is not uniform. It is a best-effort attempt to query as fast as possible. For uniform sampling, use PmacTerminalGather.

Motor positions, velocities, or following errors can be displayed. The chart control palette allows panning and zooming of the display. Use the **Free Run/Pause** button to pause the chart update. The **Clear** butoon will clear the chart and restart the chart history buffer. The **Save** button will prompt for a file to save the chart data as a tab-delimited spreadsheet file. Select **File»Print Window** from the menu bar to print the panel or a file at any time. Servicing PmacTerminalMenu may cause noticeable glitches in the plot data. It can be disabled by clicking the **Disable Menu Track** button.



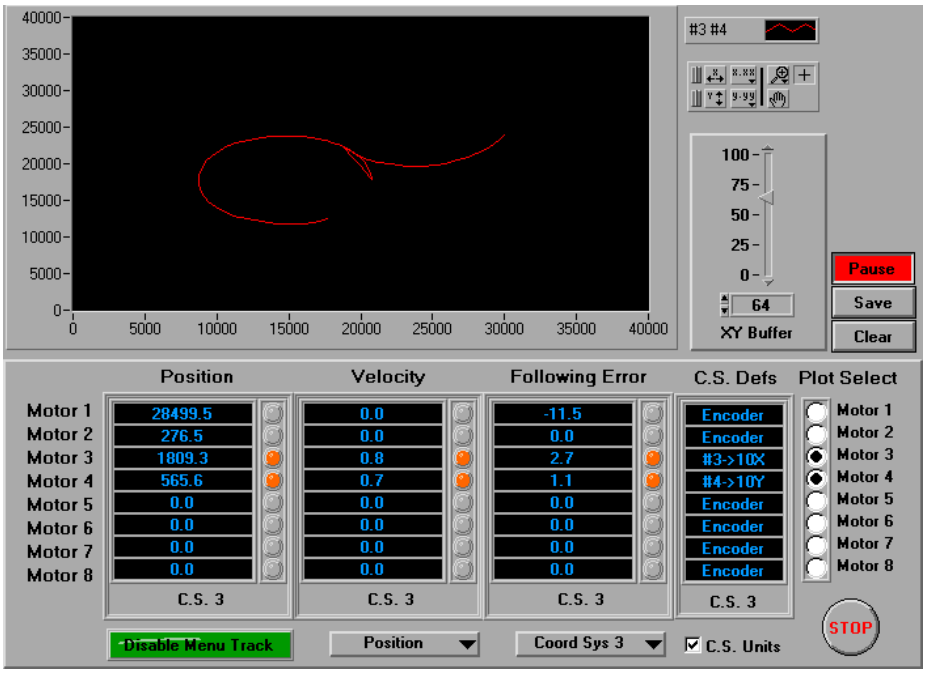
Depending on requirements, alter chart operation using the attributes available with the right mouse button. These include the length of the history axis, auto scaling of the Y-axis (P, V, or E), plot style, and many other items. To change the range on the Y-Axis click the minimum or maximum value on the axis, enter a new value and disable the auto scaling. Do not forget to save these changes to make them permanent.

The diagram has a case structure to allow selection of positions, velocities, and following errors. To plot a mix of these, modify the diagram. Pause is implemented with a case structure that prevents the update of the chart. The VIs for querying position, etc. and the PmacMotorsPVE clusters can be moved into the pause case. This way the indicator update will also pause. Save and Clear use the chart attribute nodes to access the History data required to implement their operations. These pieces of the diagram are useful in many other applications because their use is not obvious in the LabVIEW documentation.



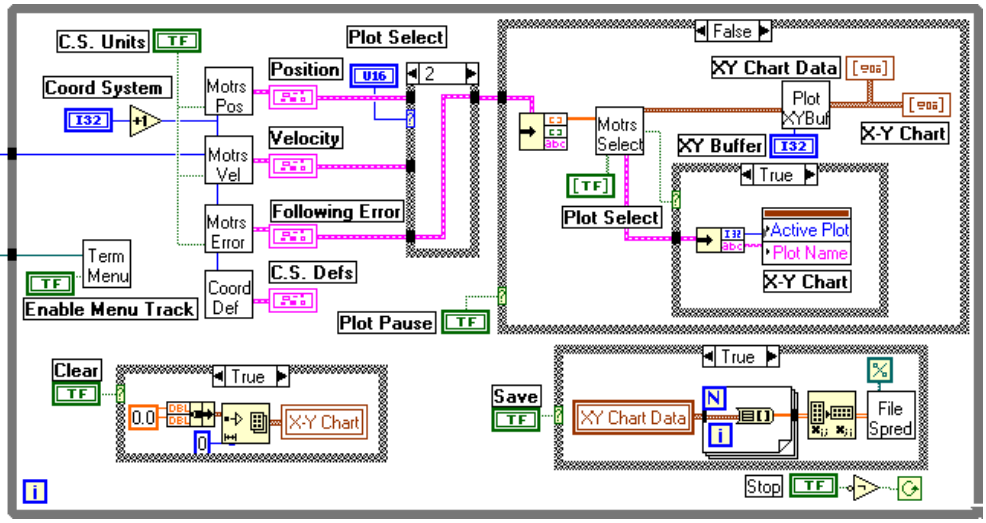
PmacTerminalMotorX-Y

This tool is based on PmacTerminalMotors. Its purpose is to monitor and plot selected motor motions executing in a 2-Axis Cartesian coordinate system. Ninety-five percent of this tool functions as introduced in PmacTerminalMotors. See the documentation for this tool. The difference is that motors selected in Plot Select are displayed in the pseudo real-time X-Y chart. To display 2-Axis data, select two motors.



The charting of 2-Axis motion uses a special chart buffer PmacMotorsPlotXYChartBuffer to fool the standard LabVIEW X-Y plot control into behaving like a real-time chart. This is required because LabVIEW's X-Y plot does not maintain a history buffer - it plots sets of points. The slider on the right of the plot specifies the length of the history buffer for PmacMotorsPlotXYChartBuffer. Like a standard real-time chart, 2-Axis position history is displayed like a snake moving around the coordinate space. The head is current position and the tail is the last position in the buffer. This tool will definitely help in understanding what moves are executing in a 2-Axis system.

The implementation of the X-Y charting is similar to that used to implement the standard chart in PmacTerminalMotors and PmacTutor10. The main difference is that the chart has no history buffer. Hence, a hidden control for the data provided to the chart named XY Chart Data has been created.



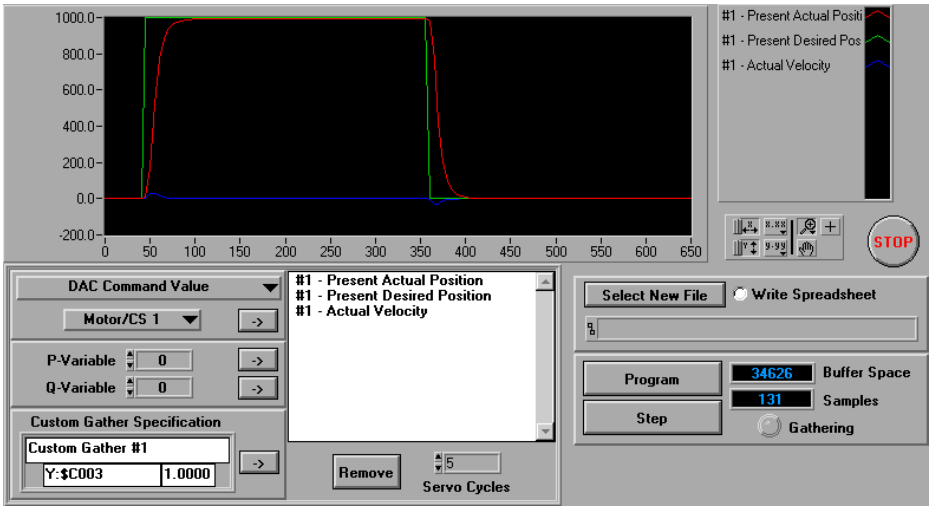
PmacPlotXYChartBuffer takes the two point X-Y cluster terminal from PmacMotorsPlotSelect VI and buffers them to generate an array of clusters for the X-Y Chart. To update the plot legend, create an attribute node for the chart, select the two items shown, and set them from the cluster provided by PmacMotorsPlotSelect.

PmacTerminalGather

This tool is a general purpose tool for synchronously gathering and plotting PMAC data and outputting the data to TAB delimited spreadsheet files for use by Microsoft Excel, LabVIEW, Matlab, or other analysis and plotting applications. It executes a step or an encapsulated motion program. Encapsulated motion programs created by PmacTerminalEdit can be installed easily in the tool by replacing the existing PmacProgSubVI with a custom one.

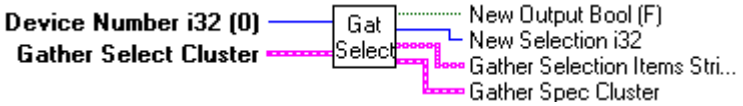
The LabVIEW chart controls enable panning and zooming of the plot. Select **File»Print Window** from the menu bar to print the panel or a file at any time.

As with the PmacTerminalMotors and PmacTerminalMotorsXY tools, the plot attributes can be modified prior to running the tool to accommodate an expected scale. PMAC Panel Pro has utilities to set the legend and plot colors to reflect the gather selections. One issue to consider when gathering data is that the gathered variables can and will, as in the sample panel, have widely varying ranges. Choose to stack the plots if necessary.



To select Address Items use the controls contained in the PmacGatherSelect cluster on the bottom left. PMAC allows selection of up to 24 Address Items. The following section covers methods for extending the PmacAddress selection VIs. The PmacGatherSelect VI builds an array of Address Items containing a string description, an address, and a scale factor using the interactive commands from the panel cluster.

- PmacGatherSelect – Maintains a PmacGatherSelect Cluster and builds a PmacGatherSpec cluster to define gather operations. A list can be built in four ways. Select an item and Motor/CS number, P-Variable, Q-Variable, or define a Custom Gather Specification. Click the associated -> button to add the item to the list on the right. The gather sample rate is defined as a number of Servo Cycles. All items are gathered at the same sample rate. Items selected in the list can be deleted using the **Remove** button.
- New Output is true when an item is added to the list with a -> button or removed from the list with the **Remove** button. New selection identifies the selected item in the gather list. Gather Selection Items String Array define the contents of the gather list. Gather Spec Cluster is an internal data type used by other PmacGather VIs to setup PMAC and collect the gathered data.



There are several methods for specifying Address Items provided by this VI and its control/indicator cluster.

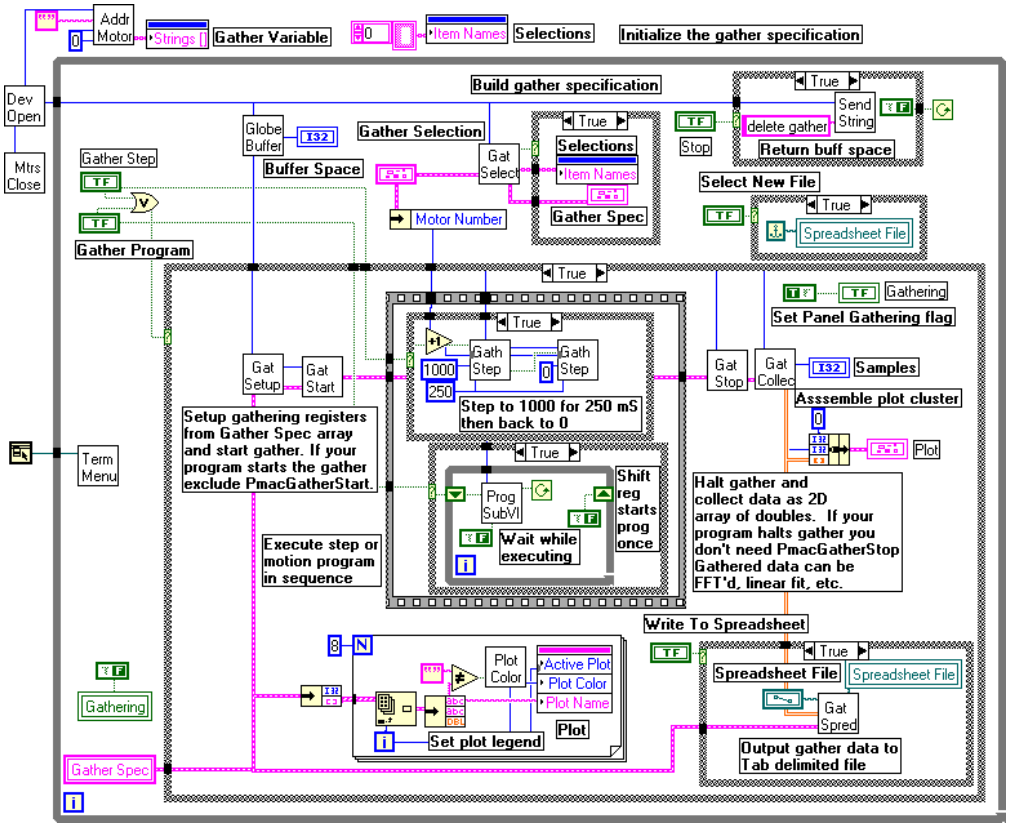
- The top group of two Menu rings allows the selection of one of 29 standard motor or CS variables and a motor or CS. Using the -> button in this group, add the selection to the Text ring on the right.
- If a P-Variable or Q-Variable number is specified and the appropriate -> button is clicked, the specified variable is added to the selection list.
- On the bottom is a Custom Gather Specification cluster that allows the entering of a description, address, and scale factor. Clicking its associated -> button adds this item to the selection list. Be aware that the PComm 32 Pro library requires addresses to be specified as:

```
X:$****
Y:$****
DP:$****
```

To write the gathered data to a spreadsheet file click the **Write Spreadsheet** radio button before executing the gather. To actually gather the data, click **Step** or **Program**. **Step** executes a 250 mS step to 1000 counts and back to 0 counts using the motor specified in the menu ring used to setup the gather. **Program** executes the encapsulated motion program wired into the diagram. During the gather, the Gathering indicator is green. After the gather, the data is plotted and the indicator turns gray. If Write Spreadsheet was selected, the system will prompt for a file name.

The diagram for implementing this tool is a great source of ideas for building gathering into applications. Being a full application tool it requires a bit more work and it is not as bad as it looks.

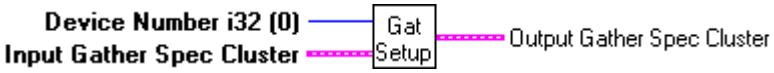
Outside the execution loop, all motor servo loops are closed by PmacMotorsCloseLoop. To exit the loop the **Stop** button first sends a delete gather command to PMAC to free the buffer space. Buffer space is monitored using PmacGlobalBuffers and an indicator. How a Gather Selection list is built is outlined in the next section.



Data is actually gathered in this tool when either the **Gather Program** or the **Gather Step** buttons are clicked. They require the Gather Spec Cluster created by the program or generated by PmacGatherSelect. The gather process begins by executing in sequence the following five VIs. In the diagram, the sequence of operations is located in a case structure.

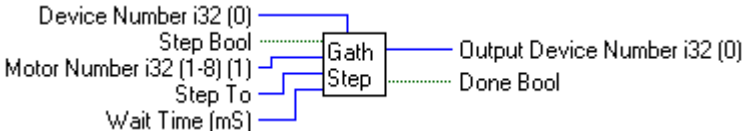
- PmacGatherSetup - Use the information in Input Gather Spec Cluster to setup a gather operation on PMAC. Output Gather Spec Cluster should be wired to PmacGatherStart, PmacGatherStop, and PmacGatherCollect to sequence operations and so that they can get the information that they require for their operation.

  The actual setup can also be done using Pewin 32 Pro, PmacTerminal, or the motion/PLC programs. This is not recommended if intending to use PmacGatherCollect to retrieve the gathered data. These methods require intimate knowledge of PMAC's internal architecture and are handled by this VI automatically.

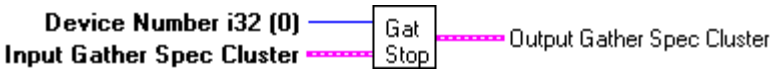


- PmacGatherStart - Start a previously defined gather operation. Start the desired motion immediately after this VI executes. Eliminate this VI if gathering is started by using the **DEFINE GATHER** PMAC program command in the program.

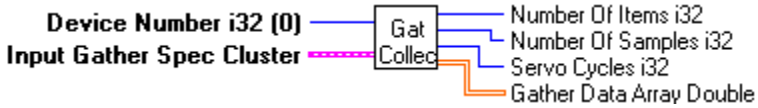
- PmacGatherStep - This is only one of any possible motion or encapsulated motion program.



- PmacGatherStop - Stop an executing gather operation. Eliminate this VI if gathering is stopped by using the **END GATHER** PMAC program command in the program.



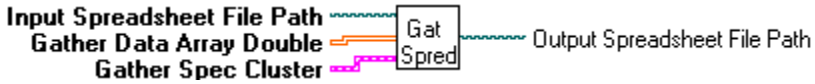
- PmacGatherCollect - Collect the gather buffer and scale the data using each items scale factor. The data in Gather Data Array is a two dimensional array of doubles with Number of Items columns and Number of Samples rows. In this format, the data can be written to a spreadsheet or processed by many different LabVIEW data analysis VIs.



The sequence frame in the middle of the case executes the step or an encapsulated motion program. Replace the motion program with a custom one or modify the entire sequence to suit individual needs.

There are two other operations performed within the main case structure. The Gather Spec cluster is unbundled and used with PmacPlotColor to setup the legend. Finally, after the data is collected it can be written to a spreadsheet if the operation was enabled prior to the gather. This is done using:

- PmacGatherSpreadsheet - Output a tab delimited spread sheet file for import into other plotting and analysis applications. If Input Spreadsheet File Path is empty of Not a Path, a dialog prompts for a file name. The file name used is provided to Output Spreadsheet File Path.



Specifying Gather Addresses

With PMAC, data can be gathered from any address. This requires an address to gather from and a scale factor to apply to the data. The PmacGather tools use a small collection of PmacAddress VIs to simplify the specification of PMAC addresses for gathering. When these tools are understood, they can be modified to suit particular needs.

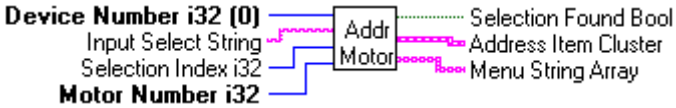
The purpose of the PmacAddress collection is to build arrays of Address Item Clusters as shown here to define an Address Item's text description, address, scale factor, and type

Address Item Cluster Specify a description, address, and scale factor for a Address Item.

- **Address Item Description** Text description of Address Item
- **Address Item Address** Address of Address Item
- **Address Item Scale** Scale factor for Address Item
- **Address Item Type** Enumerated type defining type of raw data

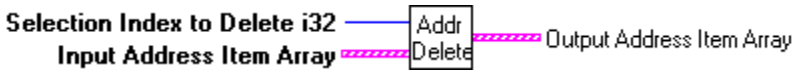
The **PmacAddress** collection contains three Vis:

- PmacAddressMotors – This VI maintains a table defining 29 of the most common Address Items. If Input Select String is the empty string, the VI produces Menu String Array describing the defined Address Items. This should be used to set the items in a Menu or Text ring control. Selection Index and Motor Number are provided by rings and define the desired item and the motor number used to compute an address for the specified item. The computed item is contained in Address Item Cluster. For a description of this computation, see the reference section and the memory map contained in the PMAC Software Reference Manual.

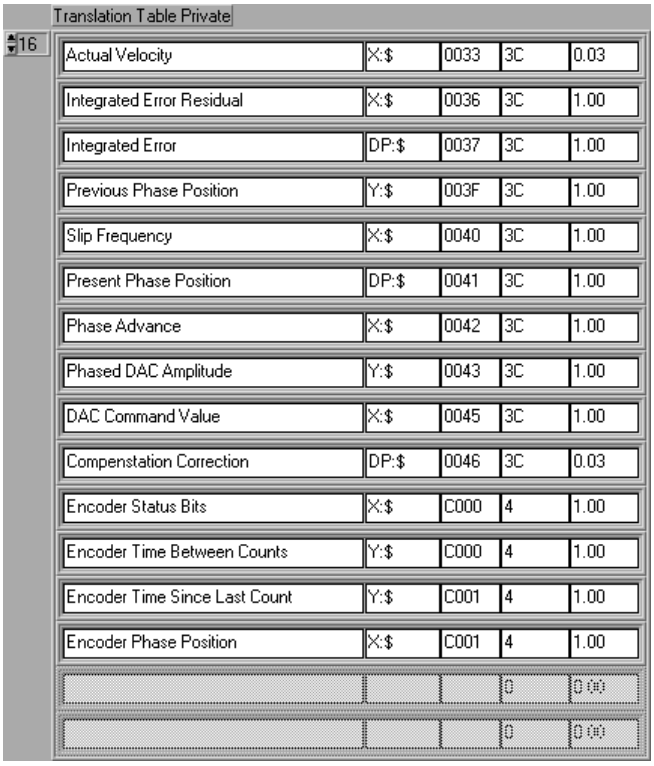


- PmacAddressAdd – Check to see if the item specified by Address Item Cluster already exists in Input Address Item Array. If it already exists, do not add it. If it does not exist, add the cluster to Output Address Item Array.

- PmacAddressDelete –Locate and remove the Address Item Cluster specified by Selection Index to Delete from the Input Address Item Array.



The PmacAddressMotors panel contains an array of clusters that define a translation table. Manually add items to this table and set them as the defaults for the control transparently adding them to the menu ring in the PmacGatherSelect control. A portion of the table provided with PMAC Panel Pro is shown here.


Translation Table Private
16
Actual Velocity X:$ 0033 3C 0.03
Integrated Error Residual X:$ 0036 3C 1.00
Integrated Error DP:$ 0037 3C 1.00
Previous Phase Position Y:$ 003F 3C 1.00
Slip Frequency X:$ 0040 3C 1.00
Present Phase Position DP:$ 0041 3C 1.00
Phase Advance X:$ 0042 3C 1.00
Phased DAC Amplitude Y:$ 0043 3C 1.00
DAC Command Value X:$ 0045 3C 1.00
Compenstation Correction DP:$ 0046 3C 0.03
Encoder Status Bits X:$ C000 4 1.00
Encoder Time Between Counts Y:$ C000 4 1.00
Encoder Time Since Last Count Y:$ C001 4 1.00
Encoder Phase Position X:$ C001 4 1.00

Each cluster item in the array consists of five items. In order from left to right these are:

- **Name** Textual description of the item to be gathered. Used in conjunction with a motor or CS number to build a unique description for plot legends and spreadsheet files.
- **Address** A string defining the size and interpretation of the data to be gathered. Legitimate designators are X:$, Y:$, and DP:$.
- **Address Offset** A hexadecimal string defining the offset address of the data to be gathered.
- **Address Stride** A hexadecimal numerical value that defines a stride to be used in computing the final gather address. The actual address is computed as (Motor Number - 1) * Address Stride + Address Offset.
- **Scale Factor** A scale factor to apply to the collected data. Some entries in this table compute this value depending on the item being gathered.

Refer to the PMAC I/O and Memory Map in the PMAC Software Reference Manual prior to modifying this table. Note that in the table the Encoder Time Between Counts has an Address Stride of $4 and an Address Offset of $C000, whereas most Address Strides are $3C. Using the values, the address for Motor 3 Encoder Time Between Counts is computed to be:

```
Y:$(C000 + (Motor Number - 1) * 4) = Y:$C008
```

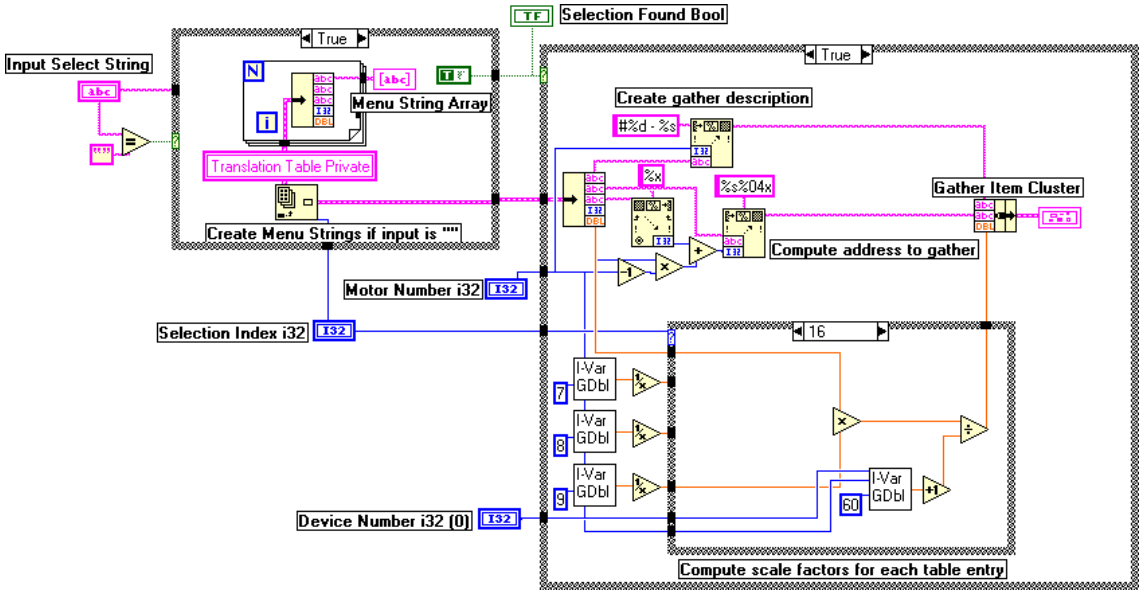
As another example, the DAC Commanded Output for motor 2 is

```
X:$(0045 + (Motor Number - 1) * 3C = X:$0072
```

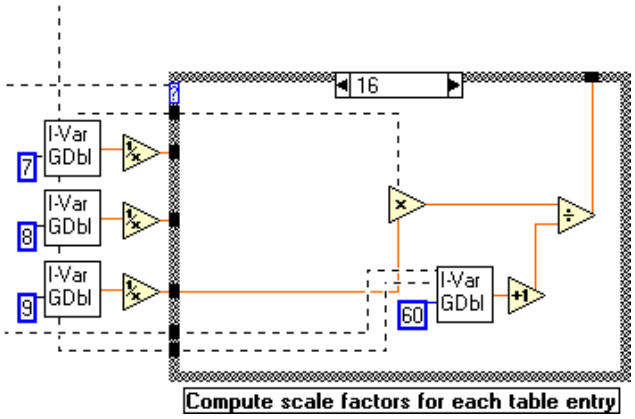
Scale Factors are a little more complex. Many Address Items in memory are scaled by one and already have the proper scaling. The most interesting ones are scaled by some combination of factors and I-Variables from internal units to encoder or coordinate system units. For example, Present Actual Position (DP:$002B) is reported in units of:

```
1 / (Ix08 * 32) counts
```

To make life easier, PmacAddressMotors computes this scale factor when building the Address Item Cluster. If items are added to the table, remember to make them the default for the table and save the VI. When an item is added, add a little wiring to the diagram for PmacAddressMotors shown here.



At the bottom of the diagram is a labeled case structure called Compute Scale Factors for each table entry. It is reproduced here for clarity.



Select the last case in the structure using the mouse and add another case. The default last case is 29 so add case 30 or whatever is needed. If the scale factor is fixed and specified in the table, wire the orange scale factor input tunnel from the unbundled cluster to the output select tunnel on the right. This will copy the scale factor in the translation table into the scale factor item for the Address Item Cluster being built. The case shown here depicts the scale factor computation for item 16 - Actual Velocity at Address Offset X:$0033 with stride $3C. The units of the gathered data, as documented in the PMAC Software Reference Manual Chapter 8, are:

```
1 / (Ix09 * 32) counts / (Ix60 + 1) servo interrupts
```

The scale factor in the translation table is 1/32. Hence, this portion of the diagram computes:

```
ScaleFactor * (1 / Ix09) / (Ix60 + 1)
```

When modifying the tables make sure that to keep an original copy of the PmacAddressMotors VI. If a control is pulled out of the cluster or the cluster is pulled out of the array, the table clears all entries.

ENCAPSULATED MOTION PROGRAMS AND PQMS

This chapter introduces a variety of VIs and tools to seamlessly integrate PMAC motion programs into the PMAC Panel Pro application. Earlier the PmacTerminalEdit was introduced. This tool allows the development of new motion or PLC programs or the modification of existing programs and with the click of a button creation of a VI wrapper for the program. In this chapter, the details of the wrapper are covered and the PmacPQM collection of VIs are introduced. These provide an interface to directly tie controls and indicators in the application panels to motion program variables.

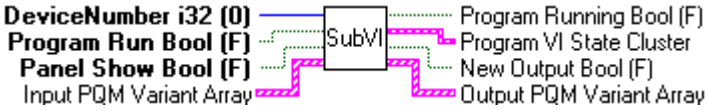
PmacProgSubVI

- PmacProgSubVI – PmacProgSubVICreate makes a copy of this VI with a new name that matches the name of a motion program. Because the motion program has the same name (with a different extension), this VI knows how to open, parse, load, and run a motion program without intervention or extra inputs. It allows the editing of the associated program and interactively executes the program. Details of its implementation are contained in the manual.

The VI downloads the associated program when first loaded, unless this option is disabled in the diagram and defaults for Program Number and Coord Number are provided for the Program VI State Cluster.

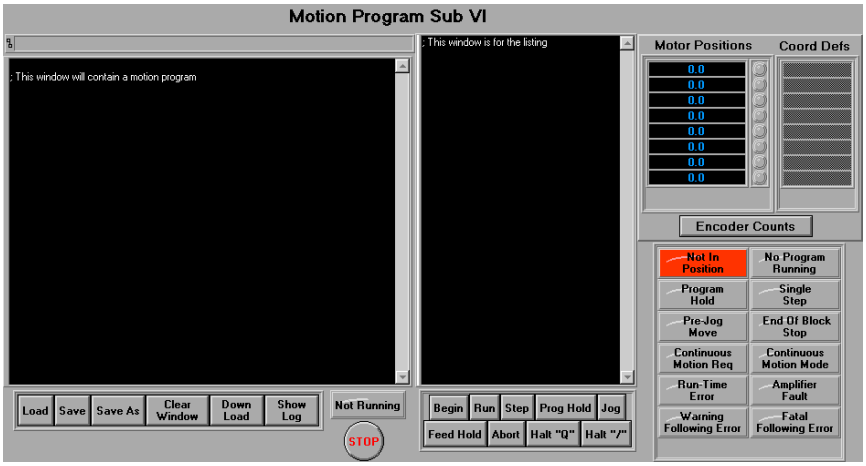
The interactive panel can be opened and used by setting Panel Show (latched) true. See the documentation for PmacTerminalEdit and PmacTerminalExecute for details on interactive execution. The panel is closed by clicking the **Stop** button on the panel

When the latched input Program Run is true, Input PQM Variant Array is sent to PMAC to initialize a program's P, Q, or M-Variables. The program is then started as long as there is no program executing in the associated CS. When Program Running is true, this or another program is executing in the associated CS.



When PmacTerminalEdit saves the motion program to a file and makes a copy of PmacProgSubVI with the same name as the motion program, the program has been encapsulated within a VI. Edit the icon of the new encapsulation or wrapper VI to represent the motion program. Use the terms encapsulation and wrapper interchangeably. Before looking at how to use the encapsulation VI, look at the new VIs panel and diagram.

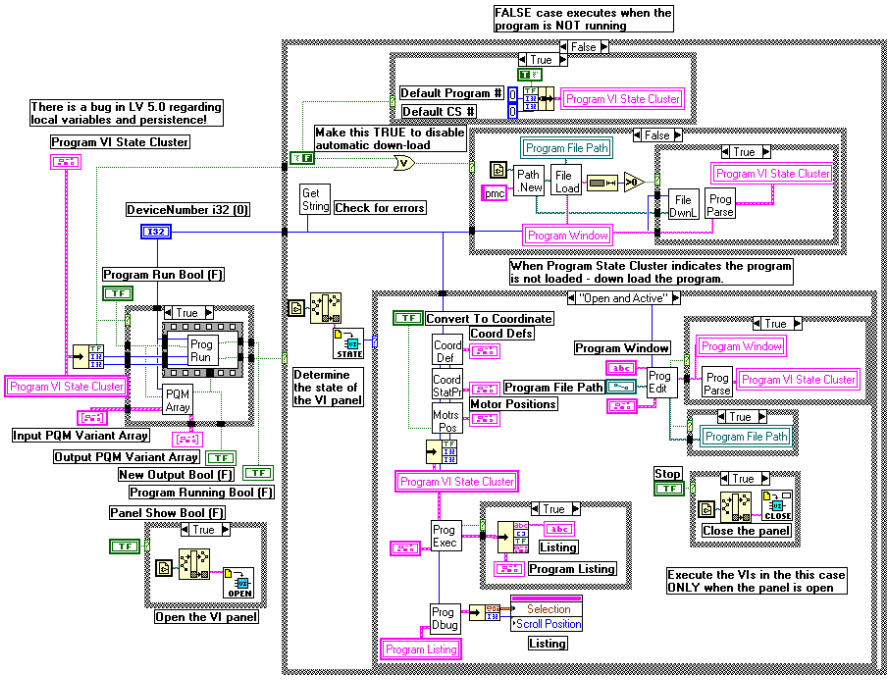
If the new encapsulation VI is opened, note that the panel is a combination of PmacTerminalEdit and PmacTerminalExecute with most of their capabilities. The purpose of this panel is to allow the editing of the associated motion program and monitoring of its execution. There is no need for buffer control or for selecting a motion program or coordinate system because these are already known by the encapsulation VI. Refer to the documentation in in this manual for details on using the capabilities of the two panel components. **Help»Show Help** will also provide detailed descriptions of the buttons and indicators.



The implementation of the VI is quite a bit different from most of those already introduced. This VI is embedded in the application's execution loop so that it can continuously monitor the attached motion program. As such, it is not wrapped in its own loop. It utilizes several VI control and server concepts found in LabVIEW to control the display of the panel and selective execution of some of its components so that it does not consume a lot of execution time unless required.

There are four major pieces of the diagram.

1. On the far left is a case structure that controls and monitors the actual execution of the attached motion program.
2. Below this is a small case structure that opens and displays the panel in response to the Boolean input Panel Show.
3. The very large case structure on the right is executed only when the program is not running. Within this case, there are two major operations. The top case structure checks the Program VI State Cluster, down loads the motion program the first time the VI is executed, and parses the program for a program number and coordinate system. This is why it is not necessary to keep track of the program number of its CS.
4. The large case structure below executes only when the panel is Open and Active and enables status monitoring, editing, and interactive execution from the panel.


FALSE case executes when the program is NOT running
There is a bug in LV 5.0 regarding local variables and persistence!
Program VI State Cluster
DeviceNumber i32 (0)
Program Run Bool (F)
Input PQM Variant Array
Output PQM Variant Array
New Output Bool (F)
Program Running Bool (F)
Panel Show Bool (F)
Open the VI panel
Default Program #
Default CS #
Make this TRUE to disable automatic down-load
Get String
Check for errors
Program File Path
Program Window
When Program State Cluster indicates the program is not loaded - down load the program.
Determine the state of the VI panel
"Open and Active"
Convert To Coordinate
Coord Defs
Motor Positions
Listing
Program Listing
Selection
Scroll Position
Close the panel
Execute the VIs in the this case ONLY when the panel is open

To hide many nasty details, the VI maintains a Program VI State Cluster. When the VI executes the first time the Program Loaded item in the cluster is false. Hence the program execution case on the left cannot execute, the large case does and the program is downloaded, thereby updating the Program VI State Cluster indicating that the program is loaded, the program number, and associated coordinate system. At this point, the VI is knows everything it needs to run and monitor the program.

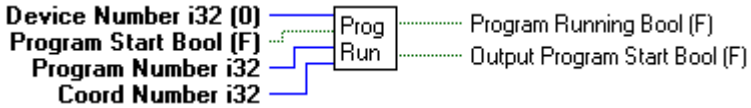
If the application does not need to be downloaded every time the system is turned on, changing the labeled Boolean constant on the top to true with the mouse will disable automatic down load. This prevents the down load but does not provide the program number or coordinate system number. To provide this information, set the constants in the case containing the Default Program # and Default CS # and save the VI.

After the down load is complete, repeated executions of this VI embedded in the application loop allow the display of the interactive panel or control and monitor the execution of your motion program. The VI provides its Program VI State Cluster as an output, so there is access to this information for building tools that are more sophisticated.

If the application provides a latched true input to Panel Show, the panel is opened and activated. The sub-case within the large case can now execute and update the panel's display. This approach eliminates a fair amount of execution overhead in maintaining a panel when not displayed. The structure of this part of the diagram is obvious. The one difference is that the **Stop** button has no loop to halt. Instead, it closes the panel and allows continued execution of the other operations in the VI.

Once the program is loaded, the case on the far left can execute. This structure performs two sequential operations. First, the PmacPQMArray VI is executed. This will set specified PMAC P, Q, or M-Variables using Input PQM Variant Array when Program Run is true. If Program Run is false, the specified P, Q, or M-Variables are retrieved from PMAC and output in Output PQM Variant Array with New Output true. The second operation is to monitor the associated motion program using PmacProgRun. If Program Run is true, the program is running and can use the newly loaded P, Q, or M variables. The Program Running output will be true indicating that the program is running. If Program Run is false, no program is executing and Program Running will indicate whether another motion program is running in the associated coordinate system. PmacProgRun is covered in detail here.

- PmacProgRun - Control and monitor the execution of Program Number in Coord Number. The specified program is started when Program Start is true and no program is currently running in Coord Number. Program Running indicates that some program - maybe not Program Number - is running in Coord Number. Output Program Start is a copy of Program Start and can be used to sequence program execution with other operations.



PmacPQMExamp

The encapsulation of a motion program with a wrapper is a huge step toward integrating PMAC with LabVIEW. The PmacPQM collection of VIs carry this further by providing an architecture for tying controls and indicators to the P, Q, and M-Variables used by the PMAC motion programs and PLCs. To illustrate how to do this, use PmacPQMExamp located in the \PmacPQM directory.

To begin, look at the sample motion program PmacPQMTest.pmc that PMAC will run (.pmc is used by Delta Tau software tools to indicate a motion program). Note that the associated encapsulation VI PmacPQMTest.vi was created by PmacTerminalEdit.

PmacPQMTest.pmc

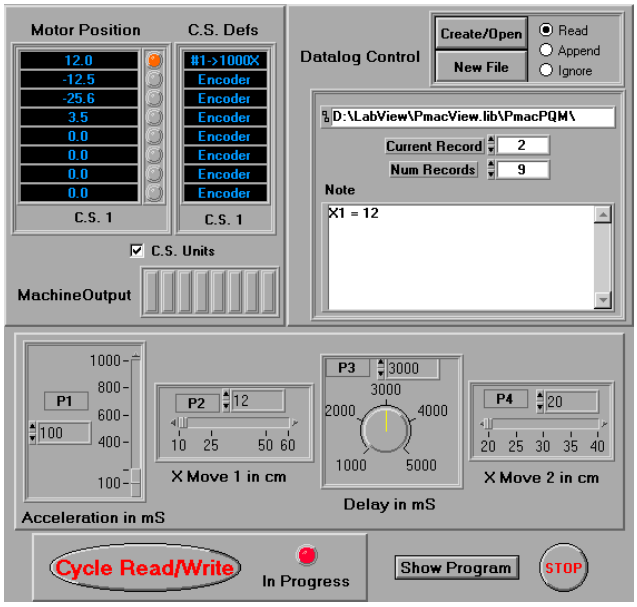
```
; USE CS &1          ; Parsed by PMAC Panel Pro during download
Close               ; Always close any open buffers
&1                  ; Define the CS
#1->1000x
m1->*               ; Redefine M1 as standard output port
m1->y:$ffc2,8,8,u
open prog 32        ; Parsed during download
clear               ; Otherwise appended to buffer!
linear              ; Set move modes
abs
ta(P1)              ; Set move - Accel time is P1
ts250
tm1500
m1 == 1             ; Show bit on port move X to P2
```

```
X(P2)
DELAY(P3)          ; Delay for P3 mS
m1 == 2            ; Update the port
X(P4)              ; Move X to position P4
DELAY1500
ta250              ;  New move parameters
ts125
tm750
m1 == 4            ; Return home
x0
dwell 100
m1 == 0
close
```

This program uses four P-Variables to define its execution and motion:

1. P1 - Acceleration time for first few moves
2. P2 - First X position
3. P3 - Delay time after move
4. P4 - Second X position

Because P, Q, and M-Variables are used to configure a motion or PLC program, PMAC Panel Pro provides a collection of VIs to take values from panel controls and set associated P, Q, and M-Variables for use by the programs. Then start program execution. The panel for the example shows a familiar set of indicators to monitor motor motions on the top left.



Below this are four PmacPQM Cluster controls associated with the four P-Variables used by the program P1 - Acceleration in mS, P2 - X Move 1 in cm, etc. Each cluster contains a control for the value of the variable and a string control specifying which P, Q, or M-Variable. At the bottom of the panel is a Cycle Read/Write button to begin execution of the encapsulated program and an In Progress indicator to monitor the execution of the program. The **Show Program** button will open the encapsulated program's interactive panel thereby allowing interactive modification of the program and stepping through its execution.

PmacPQM provides the ability to log P, Q, or M-Variables to standard LabVIEW datalog files. The logging process is controlled by the Datalog Control Cluster and Datalog Display Cluster in the upper right and is sequenced with the **Cycle Read/Write** button.

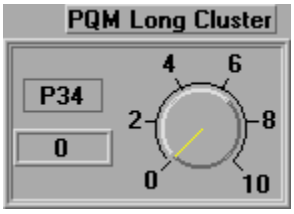
Click the **Create/Open** button, and the system prompts for the name of a datalog file. Select an existing file created using this example or provide the name for a new file. There are two example files named datalog.dat and datalog1.dat. New File closes an existing log file and allows the selection of a new one. This must be done prior to actually logging PQM data. The **Read/Append/Ignore** radio buttons define what to do with the PQM data when the **Cycle Read/Write** button is clicked.

- **Read** – It is assumed that an existing data log file created earlier was opened. Use the record specified by the Current Record to read a PQM record, set the P, Q, or M-Variables in PMAC using the retrieved record, then execute the encapsulated motion program. The values on the cluster controls will change to those read from the record when the **Cycle Read/Write** button is clicked. Using this, replay previously executed tests and configurations. The state of the panel illustrates that a Read operation was performed during the last cycle using record 1 (after the cycle Current Record was incremented to 2). The note indicates that X1 = 12 and indeed P2 has a value of 12.
- **Append** – Read the PQM cluster controls, append them to the datalog file, send them to PMAC, and start the execution of the encapsulated motion program. Add a note to the record prior to clicking the **Cycle Read/Write** button.
- **Ignore** – Keep the datalog file but do not read or write anything. Simply pass the PmacPQM clusters to PMAC.

PmacPQM Clusters

Four standard PmacPQM clusters are provided for use on the applications panels. Defining these clusters binds the P, Q, or M-Variable's name with the actual numerical value to be used with the variable. These are based on variations of the cluster definition for PmacPQMLong.

- PmacPQMLong – Cluster for tying P, Q, or M-Variable definition with an i32 control/indicator. After inserting on the panel, specify a P, Q, or M-Variable name for the Variable Item and make it the default using **Right Mouse Button»Data Operations»Make Current Value Default**. Replace Control to reflect the requirements.



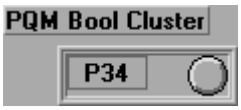
PQM Long Cluster Cluster for tying P, Q, or M-Variable definition with an i32 control/indicator. After inserting on the panel specify a P, Q, or M-Variable name for the Variable Item and make it the default using **RightMouseButton»Data Operations»Make Current Value Default**. Replace Control to reflect the requirements.

Variable String defining P, Q, or M-Variable name (e.g. P34)

Control Control for associated P, Q, or M-Variable

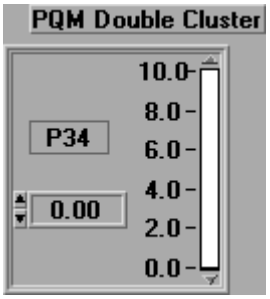
PQM Type

- PmacPQMBool



- PmacPQMShort

- PmacPQMDbl



When these are inserted on the panel, feel free to move the items around, replace the actual control, change the format and range, color, Boolean text, etc. Remember to keep the cluster order as indicated. When the name of the Variable item is defined in the cluster, it is a string (i.e. P34). Set this as the default for each control in the panel and save the VI using the cluster (not the original cluster itself).

PmacPQMVariant functions as a neutral or void type of PQM cluster.

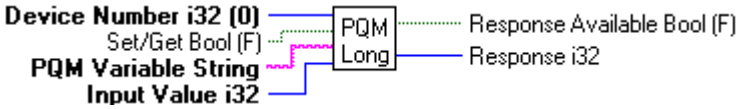
- PmacPQMVariant – Cluster for tying P, Q, or M-Variable definition with a PQM type-neutral string. Generally, this cluster is not used on application panels.

PmacPQM Conversions

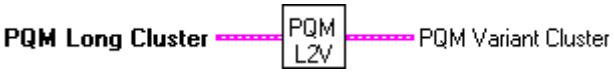
There are three types of PQM VI for processing PQM clusters. The examples given here are for the PmacPQMLong collection. Similar collections exist for PmacPQMBool, PmacPQMShort, and PmacPQMDbl.

- PmacPQMLong – If Set\Get is false or not wired, get the Long PQM Variable specified by PQM Variable String. Response Available will be true to indicate Response contains the new value. If Set\Get is true, set the Long PQM Variable using Input Value. Response Available will be false and Response defaults to Input Value.

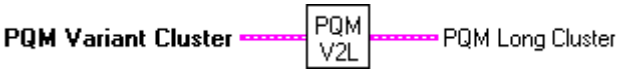
If an M-Variable is specified, it must be defined using Pewin 32 Pro, PmacTerminal, or PmacCommSendString.



- PmacPQMLong2Var – Convert the PQM Long Cluster to a type-neutral PQM Variant Cluster.

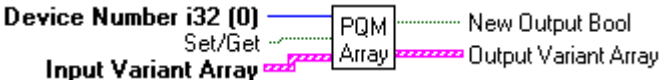


- PmacPQMVar2Long – Convert a type-neutral PQM Variant Cluster to a PQM Long Cluster.



The purpose of the 2Var and Var2 VIs is to convert clusters of specific types to and from neutral **PmacPQMVariant** types for building arrays that bundle PQM cluster controls into a single item.

- PmacPQMArray – Set or Get a collection of PQM variables as defined by an array of PmacPQMVariant clusters.



The use of arrays greatly simplifies the development of PQM configuration panels for your applications. PmacProgSubVI VIs created by PmacTerminalEdit accepts the arrays as inputs and provide them as outputs. This allows the updating of the program P, Q, or M-Variables prior to actually executing the program and monitoring any PQM variable used by the program as it executes.

PmacPQM Datalogging

PMAC Panel Pro supports data logging of PmacPQM clusters using the VIs in the PmacFile collection. These can be modified to support record attributes such as time stamps in support of your particular needs.

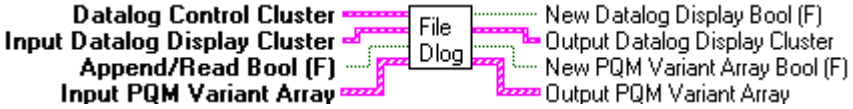
- PmacFileDatalog - Manage datalog operations for type-neutral PmacPQMVariant Arrays.

Operations as specified by the radio buttons in Datalog Control Cluster are performed when Append/Read is true. A file must be selected prior to executing the operation using the **Create/Open** button or **New File** button in the cluster. The file is opened and closed on every transaction. After an operation New Datalog Display is true and Output Datalog Display Cluster contains updated operation status for the application;'s cluster.

Append operations writes Input PQM Variant Array to the end of the file specified in Input Datalog Display Cluster and updates Current Record and Num Records in the output cluster. The contents of the Note window are appended with the record.

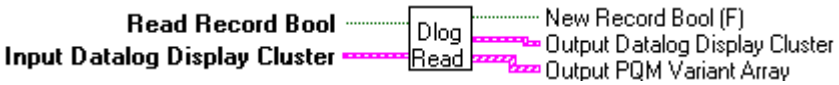
Read operations reads the record specified by Current Record in Input Datalog Display Cluster from the specified file and generates a new Output PQM Variant Array. The availability of new data is indicated by New PQM Variant Array true. Output Datalog Display Cluster increments Current Record and displays the Note, if any, attached to the record. Read operations cannot read past the end of the file and simply reads the last record in the file.

To change the data logged by this VI, simply change Input and Output PQM Variant Array to another data type. Similar modifications to PmacFileDatalogAppend, Create, and Read are also required.



- PmacFileDatalogAppend - When Append Record is true append Input PQM Variant Array to the file specified in Input Datalog Display at the end of the file. Update the Current Record and Num Records in Output Datalog Display Cluster. Indicate the new data by setting new Datalog Display true.

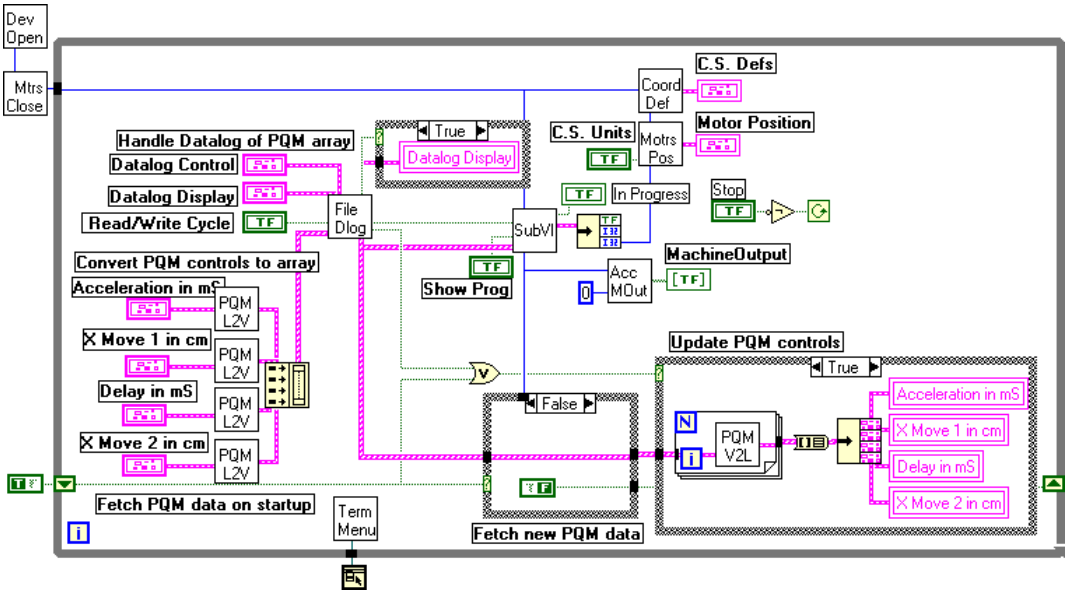
- PmacFileDatalogRead - When Read Record is true, read Output PQM Variant Array from the file specified in Input Datalog Display using Current Record. Update the increment Current Record in Output Datalog Display Cluster and display the Note, if any, stored with the record. Indicate the new data by setting new Datalog Display true.



- PmacFileDatalogCreate – Create or Open an existing datalog file to store data of type Input PQM Variant Array along with notes. When Create/Open is true, use the path in Input Datalog Cluster. If this path is empty or Not a Path is displayed, display a file selection dialog. When a file name is entered or an existing file is selected, the number of records in the file is determined. All updated information is available in Output Datalog Display Cluster and indicated by New Datalog Display true.

Using Encapsulated Motion Programs

PmacPQM ICVs can be combined with the custom PmacProgSubVI created by PmacTerminalEdit to build great applications. The diagram for the example is shown here.



The application has the standard execution loop with menu processing and a **Stop** button. In the middle is the PmacProgSubVI VI created for the motion program PmacPQMTest.pmc already introduced. When the **Cycle Read/Write** button is clicked, a sequence of operations begins. The PmacPQM clusters on the panel are translated into PmacPQMVariant clusters and bundled into an array. The array is passed to PmacFileDatalog, which appends the array to the datalog file, ignores data logging, or ignores the current input and reads a record from the specified data log file. The array is passed to the encapsulated motion program VI along with a true Boolean. The wrapper VI will down load the variables to PMAC and start the program.

If PmacFileDatalog has new PQM array data because it read the data from a datalog file or simply passed the input array through, the PQM clusters are updated with the PQM array. This is done by the two case structures in the lower right of the diagram that convert the array items to appropriate types, unbundle them, and sets the local variables for the clusters on the panel. If this is the first execution of the VI, the shift register will query PMAC for the current P, Q, M-Variables, and update the clusters.

This example program indicates its location in the program by setting bits of a standard memory mapped machine output. The output is monitored by PmacAccMachineOutput8 and used to drive an indicator on the panel. The VIs and indicators in the upper right display the coordinate system definitions and motor position. The coordinate system number for the VIs is obtained from the encapsulated motion program VI.

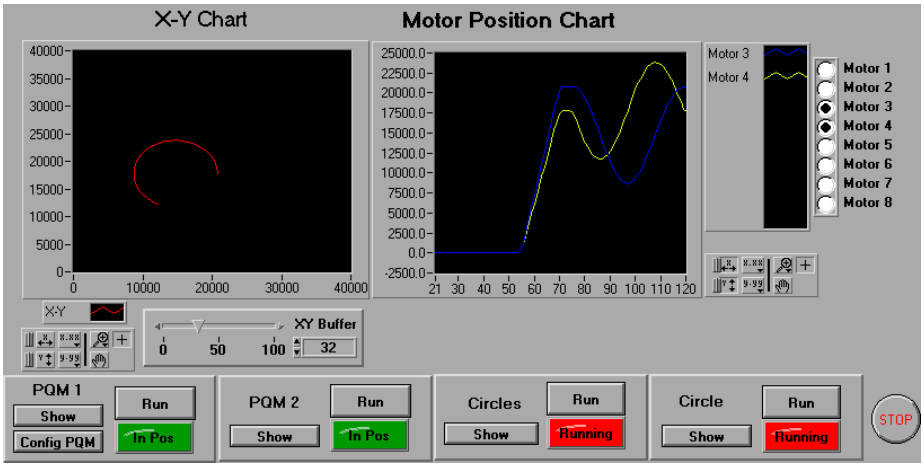
PmacTestExamp

Development of the interactive application framework can get involved. This section discusses one framework for controlling and configuring multiple motion programs.

The example PmacTestExamp, located in the \PmacTest directory, has four encapsulated motion programs.

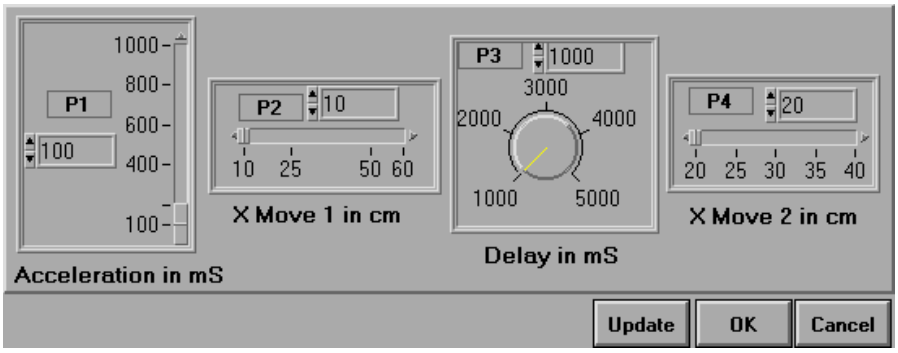
- PmacTestPQM1.pmc in CS 1
- PmacTestPQM2.pmc in CS 2
- PmacTestCircle.pmc in CS 3
- PmacTestCircles.pmc in CS 3

PmacProgSubVIs were created for each program by running PmacTerminalEdit, loading the programs one at a time, and clicking the Create Prog VI. This takes two minutes to do. PmacTestPQM1 and PmacTestPQM2 are similar to PmacPQMExamp and use P-Variables to configure their motion.



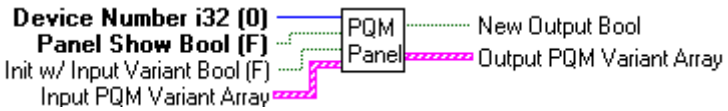
The panel for PmacTestExamp is shown above. The panel shows four sets of controls – one for each program.

- A button to start the program.
- A button to show the encapsulated motion program's execution panel.
- An LED to indicate the execution state of the program.
- PQM 1 has an extra button that, when clicked, allows modification of its associated P-Variables with the control panel shown below. The VI architecture for doing this is not really a PMAC Panel Pro design issue, but it demonstrates an approach for PQM configuration using pop-up panels in a larger application.

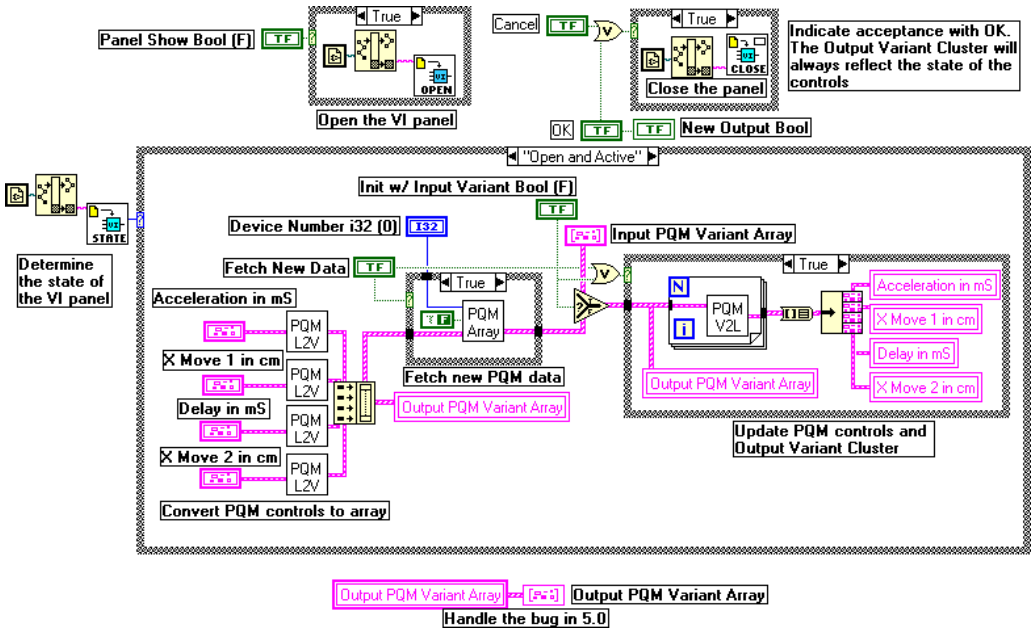


This application specific PQM configuration VI does not actually send the variables to PMAC. It creates a PmacPQMVariant array from the panel clusters that can be used by the encapsulated motion program VI in the main application. Update reads the current P-Variable values from PMAC and updates the controls on the panel. Close the panel using **Cancel** and discard the new control values. If **OK** is clicked, the VI indicates there is a new PQM Variant Array available. The disposition of the new data is up to the main application VI. The description of the VI is:

- PmacTestPQM1Panel - Group several PQM clusters together and coordinate their operation with an encapsulated motion program VI. Panel Show true displays the panel. If supplying the Input PQM Variant Array and setting the Init with Input Variant Array true, the controls initialize themselves using the array contents when the panel is displayed. If not using these inputs, first update the controls from PMAC. Output PQM Variant Array maintains any changes made using the controls from execution to execution. If **OK** is clicked, New Output Bool reflects this. Otherwise, Output PQM Variant Array contains the current state of the cluster controls.

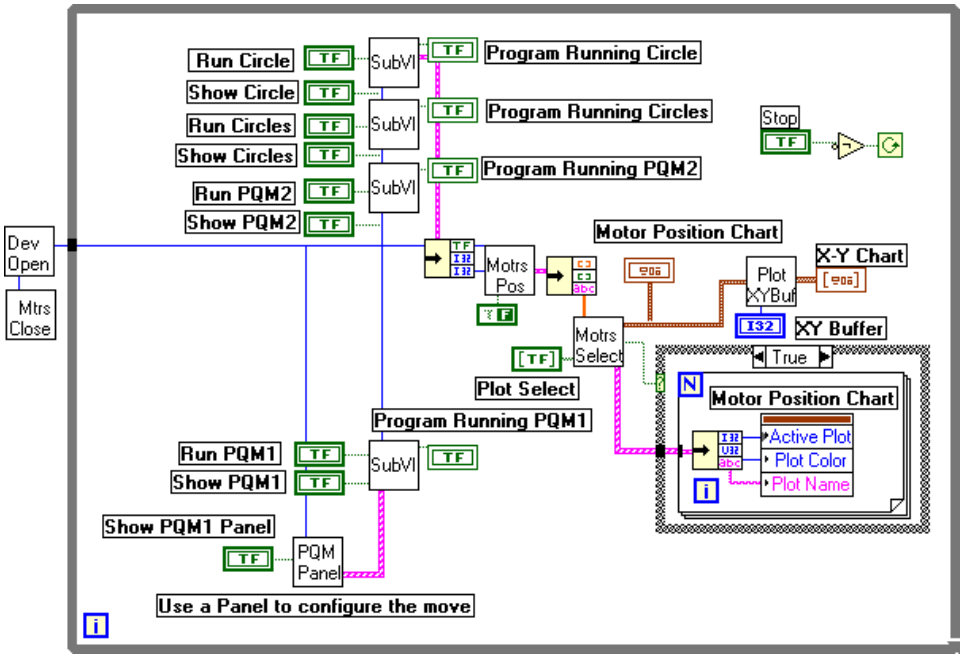


The VI forms a basis for generating custom pop-up PQM panels. The diagram shown here has pieces of PmacProgSubVI and PmacPQMExamp in it.


Panel Show Bool (F)
Open the VI panel
Cancel
Close the panel
Indicate acceptance with OK. The Output Variant Cluster will always reflect the state of the controls
OK
New Output Bool
"Open and Active"
Determine the state of the VI panel
Init w/ Input Variant Bool (F)
Device Number i32 (0)
Input PQM Variant Array
Fetch New Data
Acceleration in mS
X Move 1 in cm
Delay in mS
X Move 2 in cm
Convert PQM controls to array
Fetch new PQM data
Output PQM Variant Array
Update PQM controls and Output Variant Cluster
Handle the bug in 5.0

As with PmacProgSubVI, there is no execution loop. If Panel Show is true, the panel is opened by the case structure on the top left. If the panel is Open and Active, the large case structure is enabled and the controls on the panel are active. The panel clusters are converted to variants and used to set Output PQM Variant Array. Thus, Output PQM Variant Array always reflects the current state of the controls. If Init w/Input Variant (latched) is true or if **Update** is clicked, the controls are updated using either the newly fetched data or the Input PQM Variant Array. When **Cancel** or **OK** is clicked, the panel is closed. The state of the **OK** button is used to set New Output.

Incorporation of this pop-up VI into PmacTestExamp can be seen in the following diagram:



The handling of the Run, Show Panel, and In-Progress indicators for the individual motion programs is simple because of the encapsulation VIs. At the bottom, the **Show PQM1 Panel** button is supplied to PmacTestPQM1Panel to allow configuration of the PQM Variant Array supplied to the appropriate encapsulation VI.

The implementation of the plots, charts, and other indicators is identical to that covered already.

HOMING, ENCODERS, AND POSITION CAPTURE

PMAC utilizes a custom gate array to interface motor encoders to PMAC and perform a number of high-speed computations required to monitor motor position. When writing PMAC programs, moves in coordinate system units can be specified. Motor positions are specified in encoder counts. The gate array uses another version of encoder counts to translate motor position into encoder position.

In a previous section, the PmacMotor and PmacCoord collection of ICVs were introduced that allows the conversion between motor position in encoder counts and motor position in coordinate system units. In this section, the PmacEncoder and PmacHome collection of ICVs are introduced that give the ability to move freely between coordinate system, motor position, and encoder position specifications. These are important to relate precise position information to actions in a system. Using the encoder gate array, PMAC can be configured to:

- Capture positions in response to external or internal triggers.
- Generate triggers at pre-specified compare-equal encoder positions.

The first operation required for precision position measurement of any sort is the establishment of a zero or home position. On PMAC, this is done using an encoder capture operation that is triggered by a home position sensor. Homing details are covered in detail in the PMAC User Manual. Some of this information is repeated in this section.

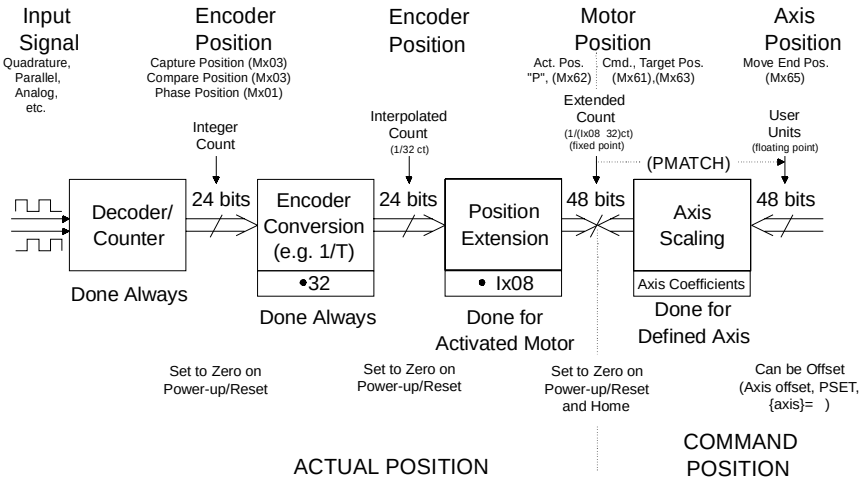
In this section, homing and position capture operations are outlined. In the next section, it is demonstrated how this same capability can be used to capture motor positions in response to external events generated by another National Instruments DAQ system or instrument. Compare operations will also be covered in the next chapter and allow data acquisition to be precisely synchronized with motion.

Position Basics

As shown below, PMAC takes position information from a 24-bit encoder register pointed to by Ix03 and extends it in software to a 48-bit register for the actual motor position. In the process of extension, it multiplies the encoder value by the position scale factor Ix08. Because the register in the encoder conversion table is in units of 1/32 of a count, the actual motor position register is in units of 1/(Ix08*32) of a count.

The extended motor position registers are set to zero on power-up and reset (unless there is an absolute position sensor), and again at the end of a homing search move. The encoder position registers are set to zero on power-up and reset only. Therefore, after a motor is homed, there is an offset between a motor's zero position and its encoder's zero position.

Try to understand this offset because the encoder registers are used for position capture and compare not the motor registers. Depending on the mechanical configuration, the rollover of encoder registers must be handled if they will be traveling more than the +/-8 million counts supported by the 24-bit encoder register. The modulo (%) operator is useful for this. For more details, refer to Synchronizing PMAC to External Events in the PMAC User Manual.



Motor position is always kept in terms of encoder counts. When a motor is assigned to an axis through a Coordinate Definition statement as in

```
&1
#1->1000X
```

for use in a motion program, the scale factor in the statement determines the units of the axis (usually inches, millimeters, degrees, etc.). As introduced in Chapter 4, programmed moves for an axis are converted to motor positions using the scale factors from the Coordinate Definition statements. It is important to realize that this conversion is for commanded positions only, and that the conversion normally goes only one-way: from axis to motor. PMAC never computes actual axis positions.

Position-Capture

PMAC's position-capture function latches the current encoder position into a special register at the time of an external trigger. The operation is set up, and later serviced, in software. The actual latching is executed in hardware, without the need for software intervention. This means that the only delays in a position capture are the hardware gate delays (less than 100 nsec), thereby providing a very accurate capture function.

Trigger Condition

The position capture register can be used both automatically, as in the firmware homing routines that handle the register directly, and manually, where the program handles the register. Manual handling of the capture register will be covered in a later section.

During motor setup, Ix25 specifies which set of flags (associated with one of the encoder counters) is used for that motor. It is important that the flag number match the position encoder number for the motor. If using ENC1 for position-loop feedback, use Flags1 (HMFL1, +/-LIM1, FAULT1), and CHC1 as the encoder index channel.

The trigger event that causes the position capture is determined by Encoder I-Variables 2 and 3 (I902 and I903 for Encoder 1). Encoder I-Variable 2 defines what combination of encoder third-channel (CHC Index channel) transition and encoder flag transition trigger the capture. If Encoder I-Variable 2 specifies the use of a flag, Encoder I-variable 3 determines which flag (usually the home flag HMFLn). Once these have been configured, the on-line **HOME** command will use the position-capture feature automatically.

Homing

Homing is a PMAC firmware function that automatically performs a number of operations to establish a motor's zero position. The homing search move can be executed with the on-line **HOME** command, from a PLC program using **COMMAND"HOME"**, or a motion program **HOME** statement. However the **HOME** command is issued, Ix23 specifies the move's speed and direction. If Ix23 is greater than zero, the homing search move will be positive. If it is less than zero, the move is negative. The acceleration for a homing search move is controlled by the same parameters -- Ix19, Ix20, and Ix21 -- as jogging moves

Action on Trigger

During the homing search move, PMAC firmware waits for the hardware trigger. When the trigger occurs, PMAC reads the position at the time of capture, usually the hardware capture register, and uses it and the Ix26 home offset parameter to compute the associated motor's new encoder zero position. Motor positions will now be referenced to a new encoder zero position plus or minus any axis offset in the axis definition statement. If the axis definition is `#1->10000X+3000`, the home position will be reported as 3000 counts.

If software over travel limits are used (Ix13, Ix14 not equal to zero), they are re-enabled at this time after having been disabled during the search for the trigger. The trajectory to the new zero position is calculated using deceleration and reversal if necessary. Note that if a software limit is too close to zero, the motor may not be able to stop and reverse before it hits the limit. The motor will stop under position control with its commanded position equal to the home position. If there is a following error, the actual position will be different by the amount of the following error.

Home Complete

If monitoring the motor from a PLC program or PMAC Panel Pro to see if it has finished the homing move, it is best to look at the home complete and desired velocity zero motor status bits. The home complete bit is set to false on power-up and reset; it is also set to false at the beginning of a homing search move, even if a previous homing search move was completed successfully. It is set to true as soon as the trigger is found in a homing search move, before the motor has come to a stop.

The home search in progress bit is simply the inverse of the home complete bit during the move: it is true until the trigger is found, then false immediately after. Therefore, the monitoring should also look for the desired velocity zero status bit to become true, which will indicate the end of the move.

Home Position Offset

PMAC stores the encoder position captured during the homing search move for the motor automatically. This value is kept in the Motor Encoder Position Offset Register [Y:$0815 (Motor 1), Y:$08D5 (Motor 2), etc.], which is set to zero on power-up/reset for motors without absolute power-on positioning. If Ix10>0 to specify an absolute power-on position read from a resolver so no homing is necessary, this register holds the negative of the power-on resolver position. In either case, it contains the difference between the encoder-counter zero position (power-on position) and the motor zero (home) position, scaled in counts.

Note:

Prior to V1.14 firmware, this value could be obtained by using the PLC program HOMOFFST.PMC, shown in the Examples section of the PMAC User Manual. Starting in V1.14, PMAC stores this value automatically.

There are two main uses for this register. First, it provides a reference for using the encoder position-capture and position-compare registers. These registers are referenced to the encoder zero position, which is the power-up position, not the home (motor zero) position. This register holds the difference between the two positions. This value should be subtracted from encoder position (usually from position capture) to get motor position, or added to motor position to get encoder position (usually for position compare).

To move an axis until a trigger is found, then convert the captured encoder position to a motor position, use the following M-variable definitions:

```
M103->X:$C003,24,S  ; ENC1 position-capture register
M117->X:$C000,17    ; ENC1 position-capture flag
M125->Y:$0815,24,S  ; #1 encoder pos offset register
```

Zero-Move Homing

To declare the current position the home position without commanding any movement, use the **HOMEZ** (on-line) or **HOMEZn** (motion program) command. These are like the **HOME** command, except that they immediately take the current commanded position as the home position. The Ix26 offset is not used with the **HOMEZ** command. This is not a reliable home and the PMAC Panel Pro VIs introduced in this section and the next do not handle this phantom home offset. Fake this by modifying PmacEncoderOffset.

Note:

If there is a following error when the **HOMEZ** command is given, the reported actual position after the **HOMEZ** command will not be exactly zero; it will be equal to the negative of the following error.

Homing into a Limit Switch

It is possible to use a limit switch as a home switch. However, first disable the limit function of the limit switch to have the move to finish normally; if this is not done, the limit function will abort the homing search move. Even so, the home position has been set; a **J=0** command can then be used to move the motor to the home position.

Note:

The polarity of the limit switches is the opposite of what is expected. The -LIMn input should be connected to the limit switch at the positive end of travel; the +LIMn input should be connected to the limit switch at the negative end of travel.

To disable the limit function of the switch, set bit 17 of variable Ix25 for the motor to 1. For example, if I125 is normally $C000 (the default), specifying the use of +/-LIM1 for motor 1 and setting I125 to $2C000 disables the limit function.

It is a good idea to use the home offset parameter Ix26 to bring the home position out of the limit switch, so the limits can be re-enabled immediately after the homing search move, without being in the limit.

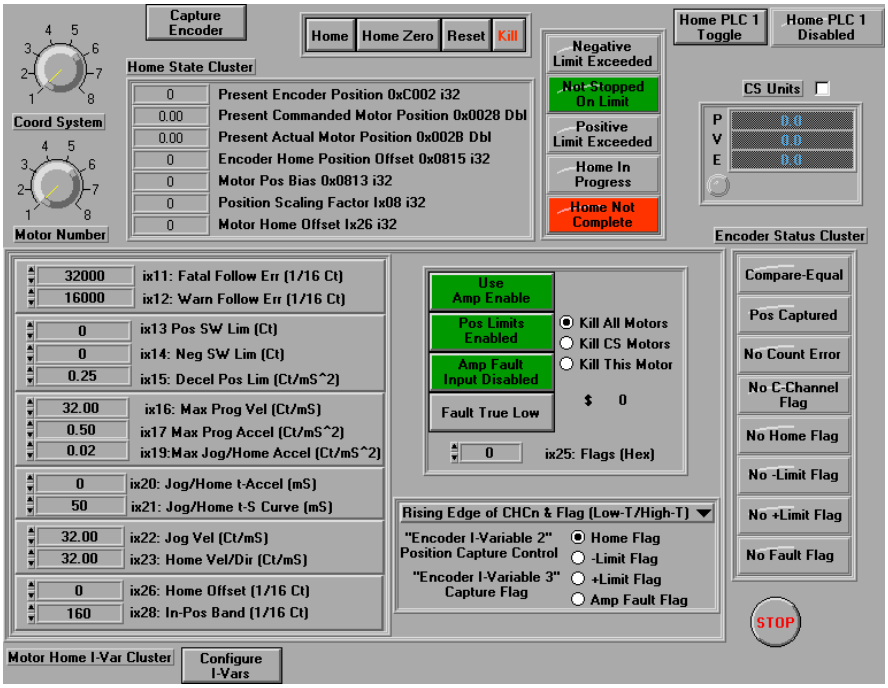
Homing from PLC and Motion Programs

The PMAC User Manual has an extensive section on homing techniques using PLC and motion programs. These are not covered in this manual. However, the programs for these are included in the PmacHome collection of ICVs.

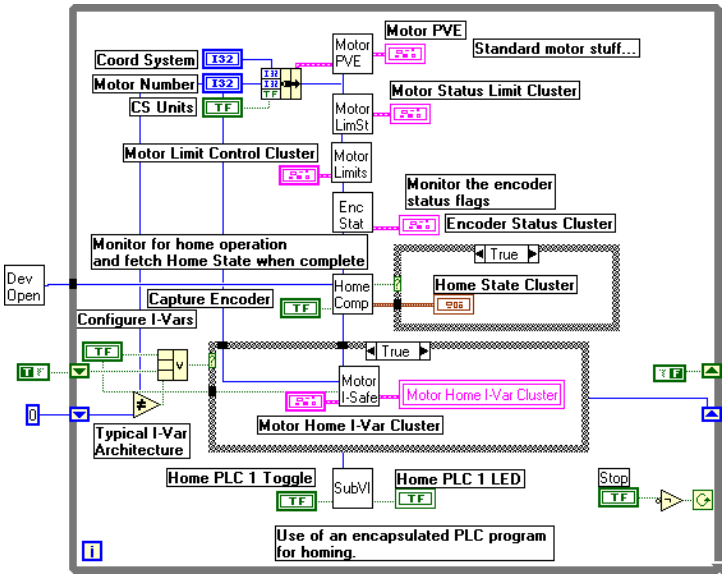
PmacHomeExamp

Having covered the basics of position capture and homing from a purely PMAC perspective, the ICVs available for use in the applications are outlined. Examine the panel for PmacHomeExamp shown below.

Many of the panel clusters should look familiar. There are three new indicator clusters associated with homing and a few new ideas associated with PLC program encapsulation. In the bottom left is a very large PmacHomeIVar cluster that borrows extensively from PmacMotorIVarSafety and PmacMotorIVarMove. It adds a new cluster for Encoder I-Vars 2 and 3. On the far right is an indicator bar that directly displays eight encoder status bits. Next to the Motor Number and Coord System knobs is a Home State Cluster. This contains data from several I-Variables and memory registers that define how motor position is transformed to encoder position. The cluster is updated any time the Capture Encoder button is clicked. In the top right is a button that will toggle the Execution State of the encapsulated PLC program that sets up and executes a homing operation.



Before going into the individual pieces of this example, look at the diagram below. As usual, the execution loop has several motor ICVs and a standard I-Variable architecture. The PmacEncoderStat VI and cluster monitor the encoder status bits and PmacHomeComplete monitors the execution of homing moves and retrieves the Home State Cluster. At the very bottom is the encapsulated PLC program Sub VI.



Configuring the Position Capture Trigger

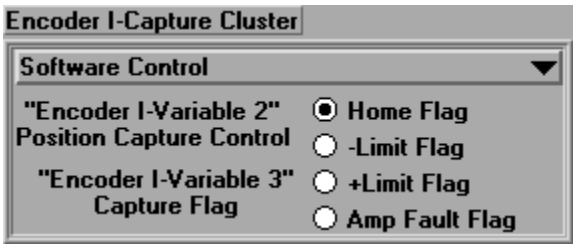
Earlier position capture and homing were discussed. Homing is a firmware operation that uses the encoder hardware position capture capabilities to establish the relationship between a motor's zero position and the encoder's zero position. To perform a homing operation, three things are required:

1. One of PMAC's four hardware trigger flags HOME, -LIM, +LIM, or FAULT must be selected.
2. A trigger condition specifying the rising or falling edge of the flag possibly combined with the encoder index channel C must be selected.

3. If a FAULT or +/-LIM flag is used, the limit or amplifier disable capabilities must be disabled. If using these, PMAC Panel Pro can perform the necessary steps, but the PMAC User Manual must be consulted for details.

   The PmacHomeIVar cluster on the example panel handles the configuration of Ix25 and encoder I-Variables 2 and 3. PmacMotorIVarFlag controlling Ix25, is detailed in in a previous section. The PmacEncoderIVarCapture cluster and its two sub controls are documented here.

   - PmacEncoderIVarCapture



Encoder I-Variable 2 Position Capture Control This parameter determines which signal or combination of signals (and which polarity) triggers a position capture of the counter for encoder n. If a flag input (home, limit, or fault) is used, I903 (etc.) determines which flag. Proper setup of this variable is essential for a successful home search, which depends on the position-capture function. The following settings may be used:

Setting	Meaning
0	Software Control
1	Rising edge of CHCn (third channel)
2	Rising edge of Flag n (as set by Flag Select)
3	Rising edge of [CHCn AND Flag n]
4	Software Control
5	Falling edge of CHCn (third channel)
6	Rising edge of Flag n (as set by Flag Select)
7	Rising edge of [CHCn/ AND Flag n]
8	Software Control
9	Rising edge of CHCn (third channel)
10	Falling edge of Flag n (as set by Flag Select)
11	Rising edge of [CHCn AND Flag n/]
12	Software Control
13	Falling edge of CHCn (third channel)
14	Falling edge of Flag n (as set by Flag Select)
15	Rising edge of [CHCn/ AND Flag n/]

Note that several of these values are redundant. To do a software-controlled position capture, preset this parameter to 0 or 4; when the parameter is then changed to 8 or 12, the capture is triggered (this is not of much practical use).

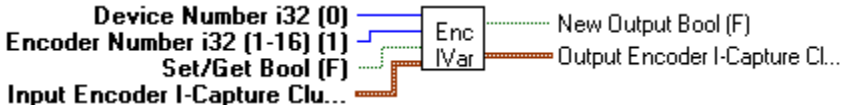
Encoder I-Variable 3 Capture Flag This parameter determines which of the Flag inputs will be used for position capture (if one is used – see I902 etc.):

Setting	Meaning
0	HMFLn (Home Flag n)
1	-LIMn (Negative Limit Signal n)
2	+LIMn (Positive Limit Signal n)
3	FAULTn (Amplifier Fault Signal n)

Typically, this parameter is set to zero, because in actual use, the +/-LIMn and FAULTn flags create other effects that usually interfere with what is trying to be accomplished by the position capture. To capture on the +/-LIMn or FAULTn flags, either disable their normal functions with Ix25, or use a channel n where none of the flags is used for the normal axis functions.

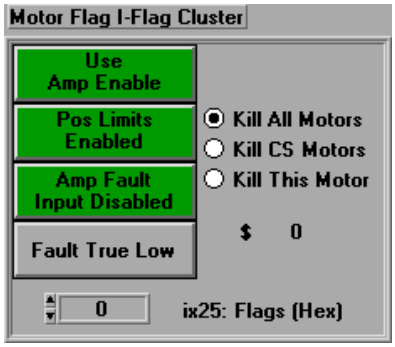
The VI for the cluster is:

- PmacEncoderIVarCapture - Follow PMAC Panel Pro's standard I-Variable VI architecture. When Set/Get is true, the Encoder I-Variables for the specified Encoder Number are set. Otherwise they are fetched from PMAC and provided by Output Encoder I-Capture Cluster with New Output true. Set/Get is not required and defaults to a Get operation.



As noted if using the +/-LIM or FAULT flags steps must be taken to modify their normal operation. These are safety limits that as a rule stop motion and disable the amplifier – not useful when trying to home the motor or setting limits. Modify this behavior using the PmacMotorIVarFlag cluster covered in a previous section.

- PmacMotorIVarFlag



By disabling the Position Limits and/or Amp Enable, home into a +/-LIM or FAULT flag. Make certain to read the PMAC User Manual section on these topics and understand what must be done. As an example of potential problems, consider this. When homing into a Limit switch, the move must be started on the proper side of the switch and moved toward it. Otherwise, it will move away from the switch and might hit a mechanical stop.

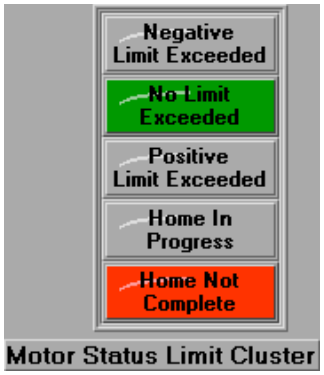
Generally, it is assumed that the PMAC is configured so that Motor N uses Encoder N and Flag N. If this is not the case, create custom VIs using the pieces provided with PMAC Panel Pro. In either situation, the architecture and examples presented here will make life a lot easier.

When the capture trigger conditions and move direction, velocity, acceleration, etc. have been configured, set these in PMAC using the **Configure I-Vars** button on the example panel.

Monitoring the Home Position Capture

Homing is a firmware operation that uses the position capture and homing move characteristics just configured. By clicking the **Home** button in the PmacMotorLimitControl cluster, the movement and monitoring of the encoder status bits can be started. When the movement starts, the Home Complete flag for the motor is set to false and the Home In Progress flag is set true. See this in the PmacMotorStatLimit cluster on the example panel.

- PmacMotorStatLimit



At a fundamental level, monitor the encoder's operation using the PmacEncoderStatFlags cluster and VI. The five indicators on the bottom of the cluster simply reflect the state of their associated inputs. Position Captured indicates that the configured trigger condition, whether used for homing or some other purpose, has occurred. Count Error is used internally by PMAC. Compare-Equal will be covered in the next section. A detailed description of these status bits and their standard PMAC M-Variable definitions follow.

- PmacEncoderStatFlags

Compare-Equal

```
M116->X:$C000,16,1      ; Compare-equals flag for encoder 1
```

This compare-equal signal is always copied into the compare-equal flag (M116 here) that is available for PMAC internal use. If using this flag internally, make sure that the signal is latched (M111=1), or it will be missed. For interrupting the host (edge-triggered), the signal should be transparent.

Position Captured

```
M103->X:$C003,0,24,S    ; Encoder 1 24-bit position capture register
M117->X:$C000,17        ; Encoder 1 position-capture flag
```

This bit goes true when the trigger condition has gone true; it goes false when the capture register is read (when M103 is used in an expression). As long as the bit is true, the capture function is disabled, read the capture register to re-enable the capture function.

Count Error

```
M118->X:$C000,18,1      ; Count error flag for encoder 1
```

If an illegal encoder transition (both channels changing on the same SCLK cycle) does get through – or around, if bypassed – the delay filter, and to the decoder, a count-error flag (M118 here) is set, noting a loss of position information.

C Channel Status
Quadrature encoders provide an index channel to indicate revolutions of the encoder. This flag is true when the channel is true.

Home Flag
A home switch may be normally open or normally closed; open is high (1 = true), and closed is low (0 = false). The polarity of the edge that causes the home position capture is programmable with Encoder I-Variables 2 and 3 (I902 and I903 for HMFL1).

+/-Limit Flags
When assigned for the dedicated uses, these signals provide important safety and accuracy functions. +LIMn and -LIMn are direction-sensitive over-travel limits that must be actively held low (sourcing current from the pins to ground) to permit motion in their direction.

The direction sense of +LIMn and -LIMn is the opposite of what many people would consider intuitive. That is, +LIMn should be placed at the negative end of travel, and -LIMn should be placed at the positive end of travel.

Fault Flag
This flag takes a signal from the amplifier so PMAC knows when the amplifier is having problems and can shut down action. The polarity is programmable with I-Variable Ix25 (I125 for motor #1) and the return signal is analog ground (AGND). FAULT1 is pin 49. With the default setup, this signal must actively be pulled low for a fault condition. In this setup, if nothing is wired into this input, PMAC will consider the motor not to be in a fault condition.

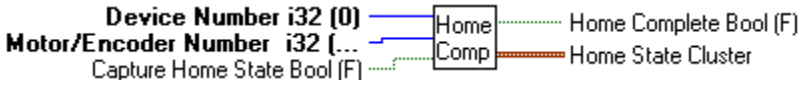
As the homing move proceeds and triggers the physical switch, the encoder will signal this using these status bits. When the configured position capture trigger condition occurs, the Position Capture bit will become true.

Home Position Transformations

Monitoring the homing operation is done in PMAC firmware. When the home move completes, the motor's zero position and its corresponding encoder's zero position will probably not match. PmacHomeComplete monitors the home operation and reports a number of I-Variables and memory registers that both demonstrate what is going on and are used for capture and compare-equal operations in the next chapter.

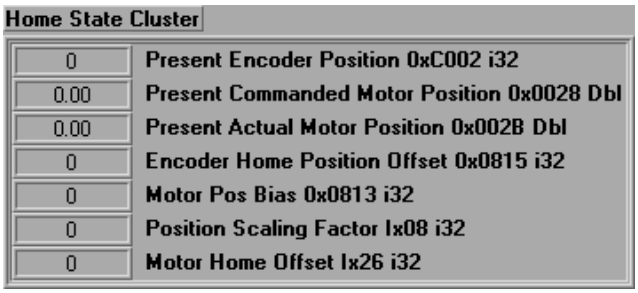
- PmacHomeComplete - Create a PmacHomeStateCluster containing I-Variables and memory registers for the specified Motor/Encoder number. The VI monitors the Home In Progress, Home Complete, and Desired Velocity Zero status bits for the motor to determine when to query PMAC for the required data. A query can also be forced if Capture Home State is true.

  This assumes Motor N uses Encoder N.



The contents of the Home State Cluster are discussed in detail here. Again, if developing PMAC Panel Pro application that uses PMAC's capture or compare-equal capabilities, these quantities must be understood:

- PmacHomeState



Present Encoder Position 0xC002 i32
The encoder Servo position register is 2 * Encoder counts with the LSB the direction bit. This output value is an actual encoder position referenced to a power-up/reset position of zero.

Present Commanded Motor Position 0x0028 Dbl
This is the motor's present commanded position in units of 1 / (32 * Ix08) encoder counts referenced to the motor's home position.

Present Actual Motor Position 0x002B Dbl
This is the motor's present actual position in units of 1 / (32 * Ix08) encoder counts referenced to the motor's home position.

Encoder Home Position Offset 0x0815 i32
This is the encoder's home offset position in encoder counts. It represents the difference between the encoder's power-up/reset zero position and the position when a home operation completes.

Motor Pos Bias 0x0813 i32
This is the position bias of the motor and represents the coordinate system translation in motor position encoder counts.

Position Scaling Factor Ix08 i32
This parameter controls how the position encoder counter is extended into the full-length register. For most purposes, this is transparent and does not need to be changed from the default.

There are two reasons to change this from the default value. First, because it is involved in the gear ratio of the position following function -- the ratio is Ix07/Ix08 -- this might be changed (usually raised) to get a more precise ratio.

The second reason to change this parameter (usually lowering it) is to prevent internal saturation at very high gains or count rates (velocity). PMAC's filter will saturate when the velocity in counts/sec multiplied by Ix08 exceeds 256M (268,435,456). This happens only in rare applications -- the count rate must exceed 2.8 million counts per second before the default value of Ix08 is a problem.

When changing this parameter, make sure the motor is killed (disabled). Otherwise, a sudden jump will occur because the internal position registers will have changed. This means that this parameter should not be changed in the middle of an application. If a real-time change in the position-following gear ratio is desired, Ix07 should be changed.

In most practical cases, Ix08 should not be set above 1000 because higher values can make the servo filter saturate too easily. If Ix08 is changed, Ix30 should be changed inversely to keep the same servo performance (e.g. if Ix08 is doubled, Ix30 should be halved).

Motor Home Offset Ix26 i32
This is the relative position of the end of the homing cycle to the position at which the home trigger was made. That is, the motor will command a stop at this distance from where it found the home flags, and call this commanded location as motor position zero.

This register permits the motor zero position to be different from the home trigger position. It is useful particularly when using over-travel limits for a home flag (offsetting out of the limit before re-enabling the flag as a limit). If large enough (greater than 1/2 times home speed times accel time), it permits a homing move without any reversal of direction.

The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts between the trigger position and the home zero position.

Example:
To make the motor zero position at 500 counts in the negative direction from the home trigger position, set Ix26 to -500 * 16 = -8000.

Encapsulated PLC Programs

In a previous section, VI wrappers that encapsulated motion programs and their operation into a single VI were introduced. PMAC Panel Pro also encapsulates PLC programs. The discussion of this topic was deferred because there was not a good example of their use – homing from a PLC program. The following PLC program, PmacHomePLC1.pmc, taken from the PMAC User Manual and uses the +LIM flag to establish a home position for motor 1.

PLC Set-up Variables (to be saved)

```
CLOSE
M133->X:$003D,13,1      ; Desired Velocity Zero bit
M145->Y:$0814,10,1      ; Home complete bit
```

PLC Program to Execute Routine

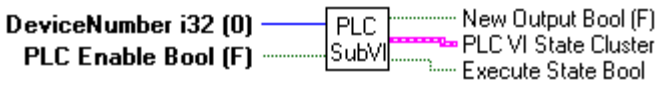
```
OPEN PLC 10 CLEAR
I123=-10    ; Home speed 10 cts/msec negative
;I125=$C000 ; Use Flags1 for Motor 1 (limits enabled)
I126=32000  ; Home offset of +2000 counts
            ; (enough to take you out of the limit)
I902=3      ; Capture on rising flag and rising index
I903=2      ; Use +LIM1 as flag (negative end switch)
I125=$2C000 ; Disable +/-LIM as limits
CMD"#1HM"   ; Home #1 into limit and offset out of it
WHILE (M145=1)    ; Waits for Home Search to start
ENDWHILE
WHILE (M133=0)    ; Waits for Home motion to complete
ENDWHILE
I125=$C000  ; Re-enable +/-LIM as limits
DIS PLC10   ; Disables PLC once Home is found
CLOSE ; End of PLC
```

Using PmacTerminalEdit, this program can be loaded. Click the **Create PLC VI** button to create an encapsulated PLC Sub VI for this PLC. This has already been done for this homing PLC, PmacHomePLC1.vi, and the other PLC and motion homing programs documented in the PMAC User Manual. The raw encapsulated PLC Sub VI is shown here:

- PmacPLCSubVI – PmacPLCSubVICreate makes a copy of this VI with a new name that matches the name of a PLC program. Because the PLC program has the same name (with a different extension), this VI knows how to open, parse, load, and run a PLC program without intervention or extra inputs. It allows interactive monitoring and changing of the PLC program's execution state. Details of its implementation are contained in the manual.

The VI downloads the associated PLC program when first loaded; unless this option is disabled in the diagram and a default for PLC Number are provided for the PLC VI State Cluster.

The VI queries PMAC for the PLC's execution state every execution. This is done whether the program is executing or not. New Output is true any time PLC Enable is true.



Using this new wrapper VI it is easy to create PLC programs and use them in the PMAC Panel Pro applications. The indicator on the example panel displays the Execute State of the properly loaded PLC program every iteration of the VI.

When clicked, the **Home PLC 1 Toggle** button on the example panel changes the state of the PLC. For the purposes of this example, click the button and the PLC begins executing and the sequence of operations in PLC 10 begin executing thereby configuring and executing the specified home operation.

There is one important point to note about this example. M133 and M145 are defined outside the actual definition of the PLC. When the VI is first executed, the entire program buffer, including these statements, is compiled and down loaded to PMAC. If certain I-Variable and memory locations are set before the **OPEN PLC** statement, these are executed when the program is downloaded, not every time the PLC is enabled.

There are a few more examples of encapsulated PLCs in the following sections.

ENCODER CAPTURE AND COMPARE OPERATION

PMAC provides sophisticated and precise motion capabilities that can be easily accessed from PMAC Panel Pro applications. When coupled with National Instruments data acquisition boards, PMAC and PMAC Panel Pro can be used to build highly integrated and precise motion based data acquisition systems using GPIB, SCXI, VXI, or DAQ boards. The degree of integration is directly related to a particular system and performance requirements.

Loosely coupled systems with slow event or clock rates that can be handled by LabVIEW can be integrated primarily with PMAC Panel Pro VIs and PMAC PLC/motion programs. Tightly coupled systems with fast clock rates or tightly synchronized motion and data acquisition requirements are handled easily using a few terminal blocks and wires to couple the hardware systems.

In this section, a set of VIs for converting between encoder position and motor position is introduced. This is followed by an example extending the position capture capabilities introduced in a previous section demonstrating how to capture positions in response to NI-DAQ signals, mechanical HARDWARE triggers, and clocks and use the captured positions in the application. Finally, PMAC's compare-equal capabilities will be introduced and several approaches for generating software and hardware triggers at specific positions while PMAC is in motion will be demonstrated. Then PMAC generated position triggers and clocks can be used by the NI-DAQ boards to control and synchronize acquisition. In all cases, PMAC Panel Pro simplifies the required tasks by working in CS units, motor position, or raw encoder units.

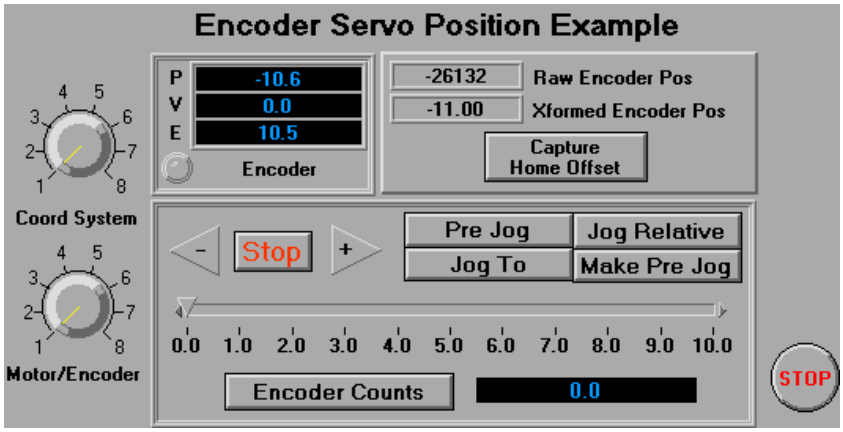
The next section outlines how to couple standard NI-DAQ boards to PMAC to synchronously trigger data acquisition at specified positions, and even use PMAC's servo clock as the DAQ sampling clock.

PmacEncoderPositionExamp

This example demonstrates how PMAC Panel Pro handles encoder positions. This is important for transforming captured positions into motor position and translating compare positions specified in motor or CS units into encoder position.

The panel shows Motor/Encoder and Coord System knobs, Motor PVE, and a Jog cluster. At the top right of the panel are two indicators that display the encoder position as the raw encoder position and the encoder position converted into motor position or CS units using VIs in the PmacEncoder collection.

Before running this VI, home the motors to be used. Do this with the example covered in a previous section or execute a home command from PmacTerminal.



Encoder Position Transformations

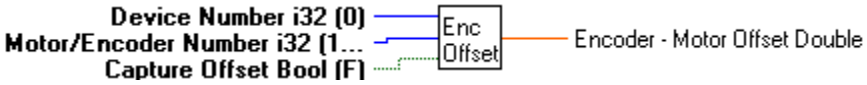
When executing this example, the position indicator in the PVE cluster will display the current motor position. The current encoder position is displayed in the indicator labeled Raw Encoder Pos. Expect these two values to be different as they are in the panel.

The most basic requirement for converting between encoder position and motor position and/or CS units is the determination of the offset between a motor's zero position and the encoder's zero position. The homing operation will generate the necessary data internally to PMAC. The following VI fetches this data and computes an offset to transform between encoder position and motor position.

- PmacEncoderOffset – Query PMAC for the encoder to motor offset captured during a home operation for Encoder/Motor Number. This assumes that encoder one is defined for motor 1, etc.

Encoder-Motor Offset provides a reference for using the encoder position-capture and position-compare registers. These registers are referenced to the encoder zero position, which is the power-up position, not the home (motor zero) position. This value is the difference between the two positions and the home offset Ix26.

This value should be subtracted from encoder position (usually from position capture) to get motor position, or added to motor position to get encoder position (usually for position compare).



The trick in using this VI is to know when to query PMAC for the Home offset information. Do this after the motor has been homed.

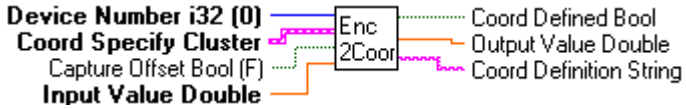
If the Capture Home Offset button in the panel is clicked, the offsets and biases for the specified motor/encoder number are retrieved. The indicator labeled Xformed Encoder Pos now displays motor position as computed directly from encoder position. There will be a very slight difference between the position in the PVE cluster and the Xformed Encoder Pos value due to the sub-count interpolation used internally by PMAC to compute motor position.

If the Encoder Counts button in the Jog cluster is clicked, the PVE cluster and the Xformed Encoder Pos indicator will both display motor position in CS units. Again, the accuracy is subject to the interpolation performed by PMAC for its own internal use.

The transformation in both cases is implemented by combining PmacEncoderOffset and PmacCoordMotorToCoord to build the following VI that converts a raw encoder position (either Capture or Compare) to motor position or CS units. Usually, this VI is used to convert a captured encoder position into motor position or CS units.

- PmacEncoderToCoord – This VI converts Input Value (Servo Position or Capture Position) from absolute encoder position to either CS units or motor position in encoder counts.

Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from encoder position to CS units. If the motor is not defined in the CS Output, Value is motor position in encoder counts. If the motor is defined and Convert is true, Coord Defined is true and Output Value is in CS units. Coord Definition is a string specifying Output Value units as Encoder or the CS definition of the motor.



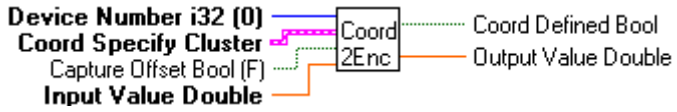
To use this VI, supply an optional Capture Offset that will fetch and compute the proper offsets. Once the offset is captured and computed, it is maintained by the VI's internal state.

PmacEncoderToCoord has a companion that takes positions specified in CS units or motor position and converts them to encoder position. This VI is most often used to take a motor position in encoder counts or CS units to encoder position for compare-equal operations.

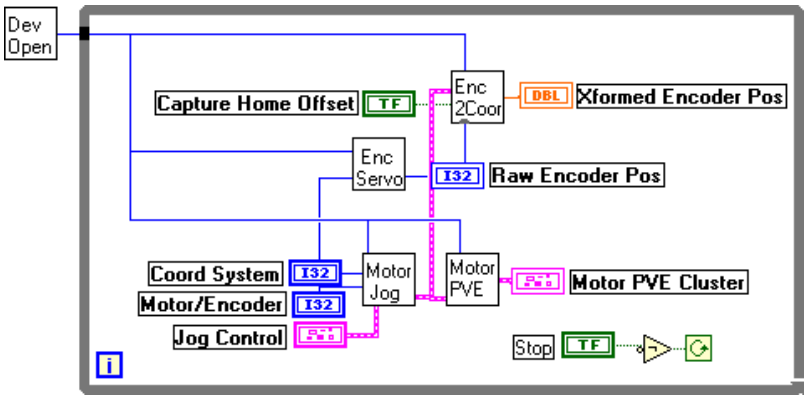
- PmacEncoderToEncoder – This VI converts Input Value in either CS units or motor position in encoder counts to an absolute encoder position for compare-equal operations.

Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Value from CS units to encoder position. If the motor is not defined in the CS Input, Value is assumed to be motor position in

encoder counts and Output Value is encoder position. If the motor is defined and Convert is true, Coord Defined is true and Output Value is scaled from CS units to encoder position.



The diagram fetches and processes the encoder position two VIs that implement the position transformations just discussed. PmacEncoderRegServo, covered later, fetches the encoder position, not capture positions, directly from the encoder, and displays it on the panel. PmacEncoderToCoord uses the transformations discussed above to compute Xformed Encoder Pos directly from this encoder position.



Position-Capture for Non-Homing Purposes

The encoder architecture and its use for homing operations was introduced earlier. Homing is a firmware command and therefore does not require monitoring of the capture flags, access the capture register, or do anything with the value. To use the position capture function for operations other than homing in the program:

- Configure the capture condition.
- Monitor the capture flag.
- Process the capture register. Do this using a PLC or using PMAC Panel Pro directly.

PLC Capture Flag Processing

If using a PLC to handle the capture operation, monitor the position-captured flag bit -- bit 17 of the encoder control/status register using:

```
M117->X:$C000,17,1
```

and the captured position using the M-Variable:

```
M103->X:$C003,0,24,S
```

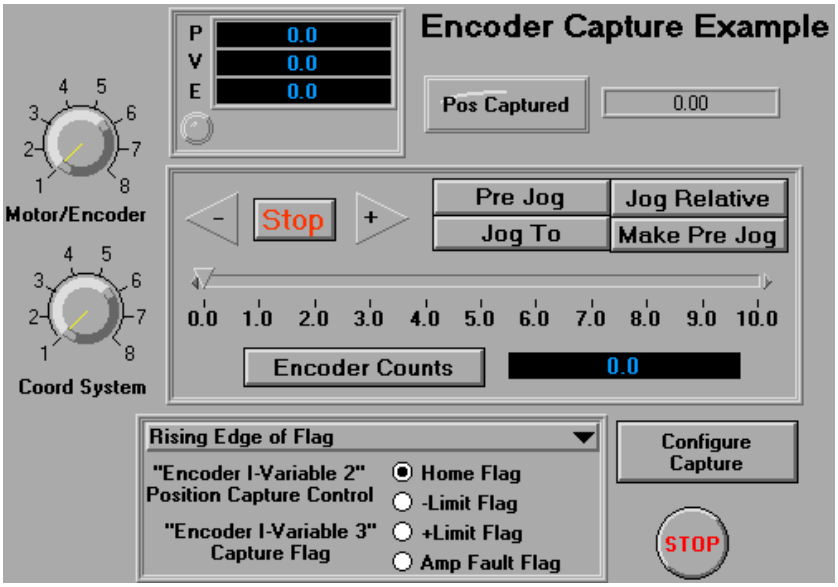
This status bit turns true when the trigger condition turns true. It returns to a non-triggered false state when the capture register (M103) is read. As long as the status bit is true, the capture function is disabled; you must read the capture register to re-enable the capture function. The example program MOVTRIG.PMC in the PMAC User Manual shows how this capability can be used for precision registration.

PmacEncoderCaptureExamp

This example demonstrates how PMAC Panel Pro handles encoder capture operations. This is important when determining the position of a motor when a trigger occurs in the system.

The panel, shown below, has Motor/Encoder and Coord System knobs, a Motor PVE indicator, and a Jog cluster. At the top right of the panel is an LED that turns Green when an externally triggered capture trigger occurs and a position indicator whose value is the position captured when the trigger occurs.

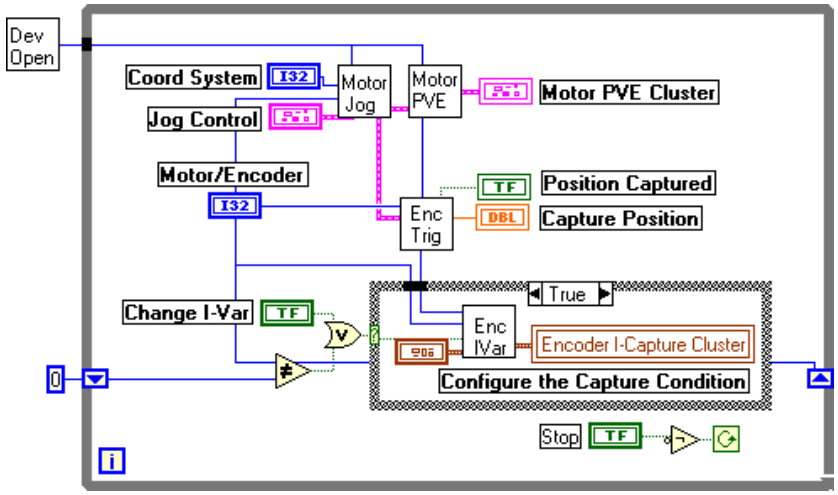
Before running this VI, home the motors to be used. Do this with the example covered in a previous section or execute a **HOME** command from PmacTerminal**.**



As with homing position capture, the trigger condition must be configured prior to use. This is done using the already introduced PmacEncoderIVarCapture cluster. The PmacHomeIVar cluster is not required because the motor has been homed and the moves being executed are not homing moves.

After selecting the capture trigger condition, the **Configure Capture** button sets the configuration in PMAC. Once this is done, the encoder is armed and waiting for the specified capture trigger.

The simplicity of the diagram demonstrates how the application is organized. At the top are the VIs to handle the jog and PVE clusters. At the bottom is the logic to handle the configuration of the capture condition. The encoder capture trigger condition is configured whenever the motor number changes or the Configure Capture button is clicked.



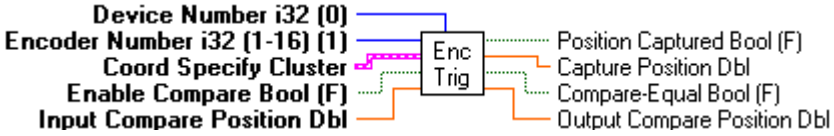
Once the capture condition is configured, PmacEncoderTrigger is used to monitor the encoder flags. When a trigger occurs the VI reads the capture register and transforms the captured position into motor position or CS units.

The VI does a lot of bookkeeping to make the job easier. To use it for capture operations, leave the Enable Compare and Input Compare Position terminals unwired. The next section of this manual will show how this VI also handles compare triggers.

- PmacEncoderTrigger – This VI maintains Encoder Number's compare-equal and capture operations and monitors the encoder's status register. Home offsets are removed or added during the processing of compare-equal and capture register data. Limitations associated with 24-bit rollover are not handled by this VI.

When Enable Compare is true, Encoder Number's compare-equal function is reset and the compare-equal register is set using Input Compare Position. This value is interpreted as being in CS units if Covert is true and Motor Number is defined in Coord Number. Otherwise, this value is interpreted as being motor position in encoder counts. Output Compare Position is a persistent copy of Input Compare Position when Enable Compare was true. The occurrence of the compare-equal condition is indicated by Compare-Equal Bool being true. This does not reset the latched condition.

When Encoder Number captures a position, Position Captured is true and the encoder's capture register is queried and converted into Capture Position in motor position encoder counts or Coord Number CS units. If Motor Number is not defined in Coord Number or Convert Bool is false, the Capture Position is motor position in encoder counts. If Convert Bool is true and Motor Number is defined in Coord Number, the value is in CS units.



External Triggers for Position Capture

Using the HOME, +/-LIM, or FAULT flags for other than their obvious purpose is very common in PMAC. It requires a simple interface circuit to be built to disconnect the physical limit or flag switches and connect the trigger signal of choice. For example, the system may have a proximity switch with TTL output to define HOME. To use the HMFL input for specified position capture operations, a TTL MUX or other form of digital selector can be used to connect the trigger signal wanted. Realize that the motor does not need to be homed using this new trigger signal. It will be used for position capture for registration or some other purpose.

In the system used to develop and demonstrate PMAC Panel Pro's capture capabilities, a TTL signal generator was used to drive a reed relay at 5 Hz and trigger the home flag. Using this configuration, the Pos Captured LED flashes on and off dutifully signaling the capture of the position. When the capture condition occurs, PMAC Panel Pro reads the capture register and converts it into motor position or CS units depending on whether the Encoder Counts button in the Jog cluster is activated. When the motor is jogging, the numeric updates with every tick of the HOME trigger signal.

PMAC Position Compare Operation

PMAC's encoder position-compare function is essentially the opposite of the position-capture function. Instead of capturing the encoder position when an external signal changes, compare operations generate a signal when the encoder reaches a specified position. In fact, the encoder register into which the captured position is written is used to store the position for compare operation. Using this capability, trigger events can be configured that occur at specified encoder positions in the system. Because the triggering is implemented in hardware, it is very fast and accurate. The next section outlines how to use external TTL level signals to trigger data acquisition on NI-DAQ boards.

Compare operations require three steps:

1. Enable and configure the operation using the encoder control register.
2. Load the compare position into the encoder register.
3. Monitor the compare-equal flag in the encoder status register and repeat these steps as required.

These steps may be performed in a PLC or a PMAC Panel Pro program.

Required M-Variables

To utilize this feature from a PLC, access the encoder control/status register and the position compare-equal register. For Encoder 1, the standard M-Variable declarations are:

```
M103->X:$C003,0,24,S    ; 24-bit pos compare register
M111->X:$C000,11,1      ; Compare flag latch control M112->X:$C000,12,1
                        ; Compare output-enable bit
M113->X:$C000,13,1      ; Compare output invert bit
M116->X:$C000,16,1      ; Compare-equals flag
```

Similar sets of registers and M-Variables are defined for the other encoder registers.

Pre-loading the Compare Position

To pre-load a compare position, assign an encoder position value to M103, such as `M103=1250`. This value must be between -8,388,608 and +8,388,607. This value cannot be read back; reading from the same address gives the position-capture register. The command can be given from a PMAC motion program, a PMAC PLC program, from the host, or using the PMAC Panel Pro VIs introduced in the following example. This is the encoder position; to reference it to motor zero position, the homing offset must be known. This translation is handled transparently by the PMAC Panel Pro ICVs in the PmacEncoder collection.

Encoder Control Bits

Three control bits configure the format of operation of the compare feature. These are software status bits and, if enabled, external hardware signals available on various PMAC cables. The flag-latch control bit (M111) controls whether the compare-equal signal is:

- **Transparent** – true only when the positions are actually equal
- **Latched** – true until actively reset by a handler

The signal is transparent if this control bit is zero and latched if the control bit is one. To clear a latched flag, take the control bit to zero then back to one.

The compare-equal signal is always copied into the compare-equal flag (M116). If monitoring this flag from a PLC or PMAC Panel Pro application, make sure that the signal is latched (M111=1), or it will be missed. To interrupt the host (edge-triggered), the signal should be transparent. PMAC Panel Pro does not currently support an interrupt driven interface. Look for this in a future release.

The output-enable bit (M112) determines whether the compare-equal flag will be output on the PMAC EQU line (1 enables). This must be set to use the signal to interrupt the host or to trigger an external event. The output-invert bit (M113) determines whether the EQU output is high true or low true (one inverts – low-true). For host-interrupt purposes, this must be configured high true.

Triggering External Action

To trigger external actions from a PMAC PC, put a connector on the E-points (E53-E65) that normally jumper these signals to the interrupt controller. An IDC 26-pin connector works nicely. These signals must be buffered; the TTL drivers for these outputs on PMAC PC are very weak. Obtain an application note on techniques for accessing these signals by contacting Delta Tau Technical Support.

On the PMAC Lite, PMAC VME and PMAC STD, a JEQU connector provides direct access to the Compare-Equals signals. The outputs are open-collector (sinking) outputs, rated to 24V and 100 mA. Replace the existing driver IC with a sourcing driver IC (UDN2981A).

To use these hardware signals, and several others, refer to the PMAC User Manual. Their use as far as NI-DAQ boards are concerned will be covered in the next section.

PLC Compare Handling

The PLC programs PmacPosCompSetup.pmc and PmacPosCompGen.pmc located in \PmacEncoder demonstrate the use of a PLC to generate a very rapid series of equals pulses at specified position intervals. PLCs are an excellent way to handle compare operations that require fast servicing. These

documented are found in the PMAC User Manual, PMAC application notes, and in the Introduction to PMAC tutorial notes. PmacPosCompSetup configures the capability by fetching the current encoder position, adding the interval, and initializing the encoder registers. This PLC is executed once to configure the operation. After configuring the operation, it starts PmacPosCompGen. This PLC monitors the encoder's compare-equal flag. When the specified position is reached, it clears the flag, loads the next compare position, and calculates the next position to be used.

PmacPosCompSetup.pmc

```
close
```

Define Encoder Registers

```
m101->x:$c001,0,24,s ; Actual position
m103->x:$c003,0,24,s ; Compare register
m105->x:$07f0,0,24,s ; Scratch register for rollover
```

Define Encoder Compare-Equal Register Control Bits

```
m111->x:$c000,11,1      ; Compare equal latch/control
m112->x:$c000,12,1      ; Compare equal output enable
m113->x:$c000,13,1      ; Compare equal output invert
m116->x:$c000,16,1      ; Compare equal flag
p101 = 50               ; Count intecrement
```

Configure the Compare Pulse

```
open plc 18
clear
```

Setup Compare-Equal

```
m105 = m101+p101  ; Save Increment + actual position
m103 = m105       ; Copy next pos into compare reg
m105 = m105 + p101    ; Update next compare position
m113 = 0              ; No invert on output bit
enable plc 19
disable plc 18
close
```

PmacPosCompGen.pmc

```
close
```

Service Routine to Service Encoder Register

```
open plc 19
clear
if (m116 = 1)
  m103 = m105     ; Update next compare position
  m105 = m105 + p101
  m111 = 0        ; Reset control bit
  m111 = 1
endif
close
```

The PLCs can be downloaded and executed using PmacTerminalEdit and PmacTerminal or, as we will show in a moment, encapsulated with a wrapper VI and controlled from an application panel.

PmacEncoderCompareExamp

This example demonstrates three methods for using PMAC Panel Pro to handle encoder compare operation. These are extremely useful for synchronizing data acquisition operations with complex motion. The three methods are:

1. Using encapsulated versions of the PmacPosCompSetup and PmacPosComGen PLC's to generate position interval clocks.
2. Directly setting an encoder compare position from the application for a one-time position-compare trigger.

3. By servicing the control, status, and position registers directly from PMAC Panel Pro.

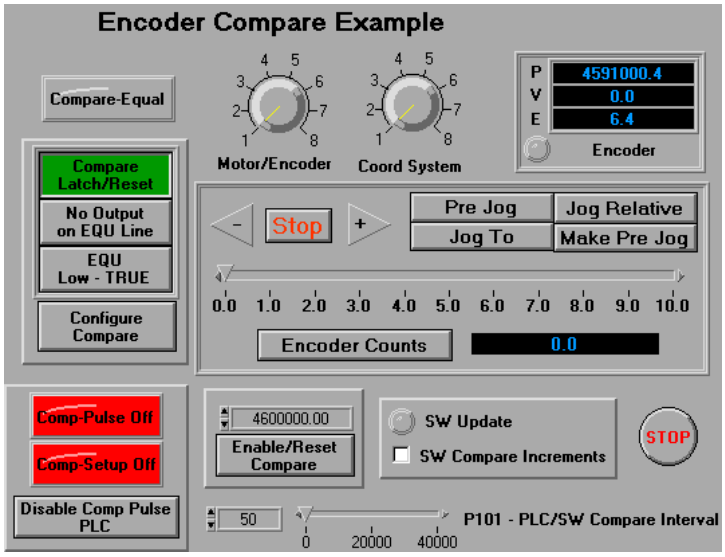
The panel, shown below, has Motor/Encoder and Coord System knobs, a Motor PVE indicator, and a Jog control cluster. At the top left of the panel is an LED that flashes green when a compare-equal condition occurs. Below this is a cluster of three buttons that allow the configuration of the encoder control bits. When these are properly set, click the **Configure Compare** button to set the encoder bits.

The remaining controls are divided into three groups — One for each method demonstrated in the example.

1. On the bottom left are LEDs indicating the Execution State of the two PLCs used to service the encoder and a button to disable the PLC handler.
2. To the right is a numeric control used to specify a compare position and a button to configure the encoder for one-time compare operation.
3. To the right of this is a checkbox that enables encoder servicing directly from PMAC Panel Pro – not the PLC. The LED indicates when a new compare position is being loaded into the encoder after a compare-equal trigger occurs.

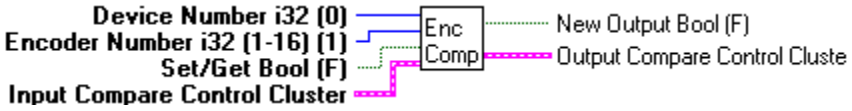
On the very bottom is a slider that specifies the interval between generated triggers. This interval is used by the PLCs and by the software interval generation.

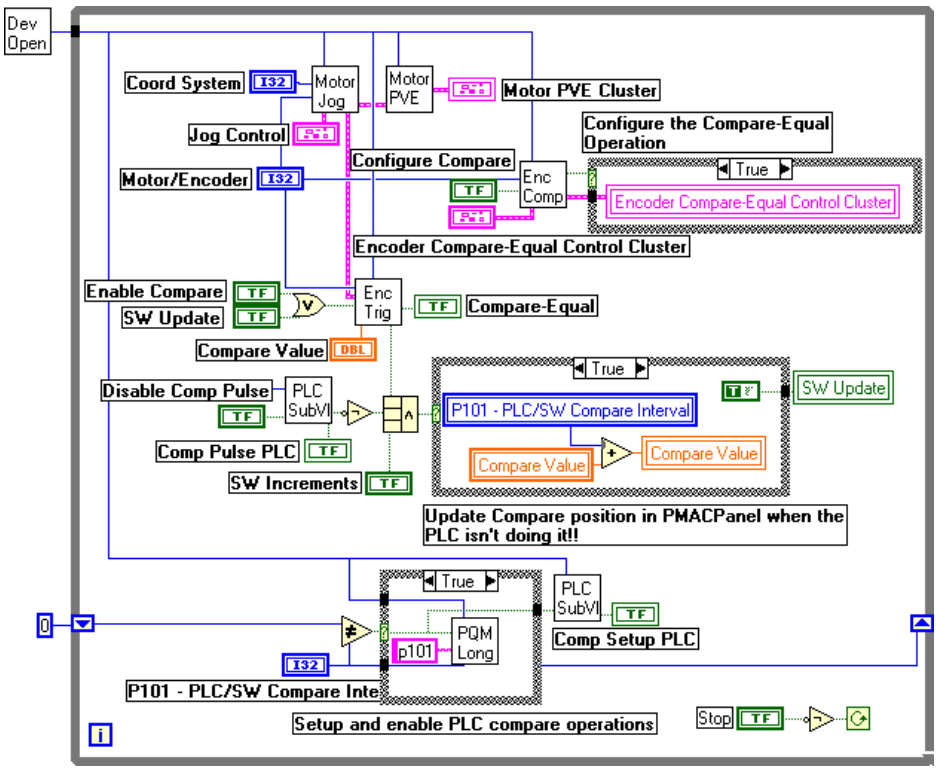
Before this VI is run, home the motors to be used. Do this with the example covered in a previous section or execute a home command from PmacTerminal.



Detailed descriptions for operating the three encoder-handling methods are covered later. At the top of the diagram are VIs to handle the Jog control and PVE indicator. Below this, PmacEncoderCompareConfig configures the encoder's compare control bits when the Configure Compare button on the panel is clicked. Configuration can be done by the PLC.

- PmacEncoderCompareConfig - Follow PMAC Panel Pro's standard I-Variable VI architecture. When Set/Get is true, the Input Compare Control bits for the specified Encoder Number are set. Otherwise they are fetched from PMAC and provided by Output Compare Control Cluster with New Output true. Set/Get is not required and defaults to a Get operation.



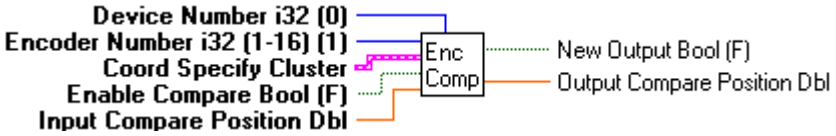


At the bottom of the diagram is logic to service the panel's P101 slider when it changes. Changes in the value update P101 and enable the encapsulated PLC PmacPosCompSetup covered earlier. Above this is the wrapper VI for PmacPosCompGen. The **Disable Comp Pulse** panel button can be used to turn the PLC on and off as desired. When this PLC is not executing, a compare trigger occurs and software increments are true, the case to the right executes and computes the next Compare Value. The final piece of the example is PmacEncoderTrigger which was covered earlier. In this example, the Enable Compare input is true whenever Enable Compare or software Update are true. This updates the encoder registers thereby arming the compare operation.

A noted earlier, PmacEncoderTrigger handles the configuration of compare operations and monitoring of capture and compare flags. The implementation of the VI is complex so it is not covered here. However, one of its pieces may be of use in the application. The following VI is used by PmacEncoderTrigger to enable and configure compare operations.

- PmacEncoderCompare - This VI reset Encoder Number's compare-equal function and set the position register using Input Compare Position when Enable Compare is true. This value is interpreted as being in CS units if Covert is true and Motor Number is defined in Coord Number. Otherwise, this value is interpreted as being motor position in encoder counts. Home offsets are removed prior to setting the encoders actual register value.

Limitations associated with 24-bit rollover are not handled by this VI. Output Compare Position is a persistent copy of Input Compare Position when Enable Compare was true.



If the application uses PLCs to handle the capture or compare triggers, do not service them with the PMAC Panel Pro application. This might create two sets of handlers for a capture or compare operation. This does not prevent the configuration of the operations using PMAC Panel Pro and servicing them with a PLC. Be aware of who is responsible for handling the encoder.

Method 1 – PLC Operation

The P101 slider specifies the interval the PLCs will use to generate compare-equal triggers. Changing this value sets P101 in PMAC and enables PmacPosCompSetup discussed earlier. This PLC captures the current encoder position, adds the interval to the position, sets the compare-equal register, and resets the compare-equal control bits. The PLC enables the PmacPosCompGen PLC and disables itself. Thus, when P101 is changed, the Comp-Setup LED briefly turns green to indicate that the setup PLC is executing. When it enables PmacPosCompGen the Comp-Pulse LED turns green to indicate that it is active, then the Comp-Setup LED turns red to indicate that it has disabled itself. This is all done using the encapsulated PLC Sub VIs.

If the motor is jogged now, the PmacPosCompGen PLC will generate compare-equal pulses every P101 encoder counts. It does this by monitoring the encoder Compare-Equal flag for the true condition, setting the next position, adding the increment for the next position, and resetting the encoder's compare-equal control bits. Because the PLC handles the flags, usually PMAC Panel Pro does not see the compare-equal condition because the PLC services the trigger so quickly. Even if PmacEncoderTrigger sees the trigger condition, it does not service it.

Bring the trigger to the external world by clicking **Output** on **EQU Line** button and then clicking **Configure Compare**. Do this before starting the Jog or enabling PmacPosCompSetup so that you do not interfere with PmacPosCompGen's handling of the encoder flags. If the external EQU signal is configured, connect an oscilloscope to the appropriate pins on JEQU or the E-Point jumpers documented in the PMAC User Manual and see the generation of the interval pulses. The next section will demonstrate how to use these pulses to synchronize the DAQ systems with the system's motion. As P101 is increased, the time between the pulses increases. When the jog is stopped, the pulse interval increases as the motor slows and eventually ceases because the motor stops. When a jog is begun, the pulse interval decreases until the motor reaches a steady state velocity.

Method 2 – One-Shot Operation

If the PmacPosCompGen PLC is disabled by clicking the **Disable Comp Pulse** PLC button, the corresponding indicator turns red. Manually configure compare-equal operations by entering a position in the numeric control above the **Enable/Reset Compare** button. The value entered, in motor position or CS units as specified by the Encoder Counts button in the Jog cluster, is used to configure a one-time compare-trigger.

Select the conditions for the operation using the buttons in the configuration cluster. The operation should be latched. Send the trigger to the external world using Output on the EQU Line and configure whether the condition is true High or true -Low. When this is complete, click **Configure Compare** then **Enable/Reset Compare**. This will configure the flags and the compare-equal value. When jog the motor is jogged, the Compare-Equal indicator will turn green when the condition occurs. The compare-equal register cannot be read, so keep track of the last value that was set. Fortunately, PmacEncoderTrigger does this.

Method 3 – PMAC Panel Pro Interval Generation

Perform the same interval generation done by the PLCs using PMAC Panel Pro. This works only when the interval rate is long relative to LabVIEW's service rate. If an interval is missed and the motor is already beyond the next interval position, the compare condition never occurs and pulses do not generate.

If the PmacPosCompGen PLC is disabled and the software Compare Interval box is checked, the trigger is handled by the 3-input **AND** case. Be careful not to enable the PLCs by changing P101 before getting the software version of the interval generation running.

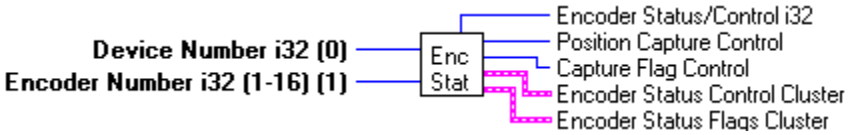
PmacEncoder Registers

Incorporating compare and capture capabilities into the on applications is facilitated by the PmacEncoder collection of ICVs. These fall into three categories: Basic encoder register access and control, conversion of encoder positions into motor position or CS units and back, and ICVs to facilitate the application.

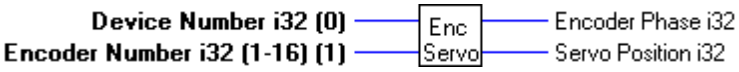
Encoder Register Access

VIs to access the encoder registers are provided but generally are not used in the applications. They are, however, useful when beginning to work with the encoders. The most important of these are:

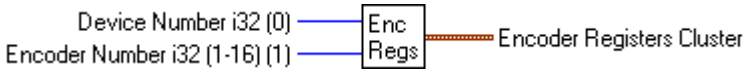
- PmacEncoderRegStat - Fetch the encoder control/status word for Encoder Number and parse it into its pieces. Encoder Status/Control i32 is the integer representation of the register. Position Capture Control can be used with PmacEncoderCaptureControl. Capture Flag Control can be used with PmacEncoderCaptureFlag. Encoder Status Control Cluster can be used with PmacEncoderStatControl. Encoder Status Flag Cluster can be used with PmacEncoderStatFlags.



- PmacEncoderRegServo - Query PMAC for the two position registers containing commutation phase and servo position. Servo Position is actual encoder position in counts referenced to a power-up/reset position of zero. Encoder Phase is used internally for commutation.



- PmacEncoderRegisters - Query PMAC for all registers for Encoder Number. Assemble the values into a PmacEncoderRegisters Cluster.



Generally, the remaining members of this collection will not be used in the application but are provided for completeness. These are:

- PmacEncoderRegTime
- PmacEncoderRegDAC
- PmacEncoderRegCapture
- PmacEncoderRegADC

PMAC AND NI-DAQ INTERFACING

In previous sections, PMAC's position capture and compare capabilities were introduced. PMAC Panel Pro provides a number of ICVs to configure, monitor, and operate these capabilities. When coupled with National Instrument's data acquisition boards, PMAC and PMAC Panel Pro can be used to build highly integrated and precise motion based data acquisition systems using GPIB, SCXI, VXI, or DAQ boards.

This section will demonstrate how to couple standard NI-DAQ boards to PMAC to synchronously trigger data acquisition at specified positions, and even use PMAC's servo clock as the DAQ sampling clock. The examples presented here in no way limit the wide array of possibilities or approaches that can be used to trigger, synchronize, and organize the motion based DAQ applications. Experience and requirements will define the approaches that best meet individual needs.

The examples of PMAC-DAQ interfacing assume that there is a basic understanding of LabVIEW's data acquisition capabilities and the acquisition boards that will be used. Similarly, there must be some understanding of PMAC's external connectors and their configuration. Connections will be made between these boards. If not certain of abilities, precisely which signals are needed, where to locate them, or what to connect them to, do not attempt to connect them. The boards, the host computers, and many other items can be damaged easily. Make certain the information presented in this section, the hardware manuals for the type of PMAC and the DAQ boards have been thoroughly reviewed. Contact Delta Tau or National Instruments Technical Support with any questions prior to proceeding.

External PMAC Signals

PMAC provides a number of hardware interfaces that can be used easily to synchronize PMAC with most NI-DAQ boards and systems. Depending on individual needs, PMAC also supports a number of I/O accessories. PMAC Panel Pro does not support all of these with ICVs. It is way beyond the scope of this manual to detail all possible approaches to accessing the interfaces on these boards. Custom ICVs can be created easily for these accessories using PmacAcc and PmacMemory ICVs.

Three primary PMAC signals will be outlined here. Position Capture was discussed in a previous section and will not be repeated here. When interfacing these signals to DAQ boards, it will be demonstrated how to use these signals to trigger acquisitions and provide the sample scan clock. Before actually using these hardware signals, consult the PMAC User Manual and the hardware manual for the particular PMAC model. The signals are:

- EQU signals
- Servo Clock
- General Purpose Machine I/O
- Position Capture Flags

PMAC's encoder clock or ADC clocks will not be considered. To use these, consult the PMAC User Manual, the hardware manual for the particular PMAC model, or contact Delta Tau Technical Support.

The following sections are reproduced from various portions of the PMAC User Manual and describe the signals, how to access them, and potential limitations in their use.

Compare-Equals Outputs (JEQU)

The compare-equals (EQU) outputs provide a signal edge when an encoder position reaches a pre-loaded value.

PMAC PC

PMAC PC does not have a dedicated connector for the EQU outputs. Instead, the signals may be accessed using a 26-pin IDC connector over E-point pairs E53-E65. The outputs are TTL-level with very low drive capability; they must be buffered externally before they can drive any real devices. Acc-27, normally used as an I/O buffer for the thumbwheel multiplexer port, can be used to drive several of these EQU lines. The 26-pin cable provided with the Acc-27 fits over the 13 jumper pairs E53-E65. Contact Delta Tau Technical Support for details.

PMAC VME

On PMAC VME, these signals are brought out on connector J7 (JEQU), referenced to digital ground (GND). As shipped from the factory, they are open-collector (sinking) outputs, with a ULN2803A driver IC, rated to 24V and 100mA each. They may be changed to open-emitter (sourcing) drivers by replacing this chip in U28 with a UDN2981A driver IC and changing jumpers E93 and E94.

PMAC Lite

On PMAC Lite, these signals are brought out on connector J8 (JEQU) optically isolated from the digital circuitry and either referenced to analog ground (AGND) or an external flag supply ground. As shipped from the factory, they are open-collector (sinking) outputs, with a ULN2803A driver IC, rated to 24V and 100mA each. They may be changed to open-emitter (sourcing) drivers by replacing this chip in U54 with a UDN2981A driver IC and changing jumpers E101 and E102.

PMAC STD

On PMAC STD, these signals are brought out on connector J6 (JEQU) on each of the piggyback boards. They are open-collector (sinking) outputs with internal 1-kΩ pull-up resistors, rated to 5V.

On PMAC STD1.5, these signals are brought out on connector J8 (JEQU), optically isolated from the digital circuitry and either referenced to analog ground (AGND) or an external flag supply ground. As shipped from the factory, they are open-collector (sinking) outputs, with a ULN2803A driver IC, rated to 24V and 100mA each. They may be changed to open-emitter (sourcing) drivers by replacing this chip in U54 with a UDN2981A driver IC and changing jumpers E101 and E102.

Servo Clock (JRS232)

PMAC's servo clock defines the rate at which servo loops are updated and background computations are performed. Using this clock for other timing is an excellent way to synchronize PMAC's movement with externally gathered data.

PMAC allows multiple cards to share a common servo clock over spare lines on the serial connector J4. The servo clock on J4 (JRS232) is located on pin 8 and is referenced to ground on pin 9. If multiple PMACs are being used, the clock signals can be shared simply by tying identical pins on the PMACs together. Acc-3D or 3L cables with extra PMAC connectors (one Acc-3E for each extra PMAC) can be used to share the clock signals in either bus or serial communications applications (and of course, for actual serial communications). In a standalone or bus-communications application, there is no need for a host drop on the cable. As is the case for the communications lines, the clock lines cannot be tied from the RS-422 port of a PMAC PC to the RS-232 port of a PMAC Lite. With the RS-422 option on the PMAC Lite (Opt. 9L), connection to a PMAC PC is possible, but the connector pinouts are different.

If serial communication is not being used, but the serial data lines are connected with the clock signals, it may be desirable to deactivate the serial port to prevent noise on the lines from creating input command characters to PMAC. On PMAC PC, PMAC Lite, and PMAC VME, this is done by setting jumpers E44-E47 ON; on PMAC STD, by making DIP switches software1-5 to software1-8 all off.

Be aware of the fact that J4 has +5VDC on pin 10.

General Purpose Digital Inputs and Outputs

The PMAC JOPTO connector (J5 on PMAC PC, -Lite, and -VME) provides eight general-purpose digital inputs and eight general-purpose digital outputs. Each input and each output has its own corresponding ground pin in the opposite row. The 34-pin connector was designed for easy interface to OPTO-22 or equivalent optically isolated I/O modules. Delta Tau's Acc-21F is a six-foot cable for this purpose.

The PMAC STD has a different form of this connector from the other versions of PMAC. Its JOPT connector (J4 on the baseboard) has 24 I/O, individually selectable in software as inputs or outputs. The rest of this discussion does not pertain to the PMAC STD port, unless specifically mentioned. Refer to the PMAC STD Hardware Reference manual for details on its JOPT port.

Standard Sinking Outputs

Caution:

Having Jumpers E1 and E2 set wrong can damage the IC.

Do not connect these outputs directly to the supply voltage or damage to the PMAC will result from excessive current draw.

PMAC is shipped standard with a ULN2803A sinking (open-collector) output IC for the eight outputs. These outputs can sink up to 100 mA, but must have a pull-up resistor to go high.

The user can provide a high-side voltage (+5 to +24V) into Pin 33 of the JOPTO connector, and allow this to pull up the outputs by connecting pins 1 and 2 of Jumper E1. Also jumper E2 must connect pins 1 and 2 for a ULN2803A sinking output.

Option for Sourcing Outputs

Caution:

Having Jumpers E1 and E2 set wrong can damage the IC.

It is possible for these outputs to be sourcing drivers by substituting a UDN2981A IC for the ULN2803A. This IC (U3 on the PMAC PC, U26 on the PMAC Lite, U33 on the PMAC VME) is socketed, and so may be replaced easily. For this driver, pull-down resistors should be used. With a UDN2981A driver IC, Jumper E1 must connect pins 2 and 3 and Jumper E2 must connect pins 2 and 3.

Input Source/Sink Control

Jumper E7 controls the configuration of the eight inputs. If it connects pins 1 and 2 (the default setting), the inputs are biased to +5V for the off state, and they must be pulled low for the on state. If E7 connects pins 2 and 3, the inputs are biased to ground for the off state, and must be pulled high for the on state. In either case, a high voltage is interpreted as a 0 by the PMAC software and a low voltage is interpreted as a 1.

Memory Mapped Access to I/O

Typically, these inputs and outputs are accessed with M-Variables. In the suggested set of M-Variable definitions, variables M1 through M8 are used to access outputs 1 through 8, respectively, and M11 through M18 to access inputs 1 through 8, respectively. This port maps into PMAC memory space at Y address $FFC2. A collection of VIs to access these are found in PmacAcc.

Synchronous M-Variables

In a motion program, when PMAC is blending or splining moves together, it must be calculating in the program ahead of the actual point of movement. This is necessary in order to be able to blend moves together and to be able to do reasonable velocity and acceleration limiting. Depending on the mode of movement, calculations can be one, two, or three moves ahead of the actual movement.

Why Synchronous M-Variables are Needed

When assigning values to variables is part of the calculation, the variables will get their new values ahead of their place in the program when looking at actual move execution. For P and Q-Variables, generally this is not a problem because they exist only to aid further motion calculations. However, for M-Variables, particularly outputs, this can be a problem because with a normal variable value assignment statement, the action will take place sooner than is expected, looking at the statement's place in the program.

For example, in the program segment:

```
X10   ; Move X-axis to 10
M1=1  ; Turn on Output 1
X20   ; Move X-axis to 20
```

It is expected that Output 1 is turned on at the time the X-axis reached position 10. Because PMAC is calculating ahead, at the beginning of the move to X10, it will have calculated already through the program to the next move, working through all program statements in between, including **M1=1**, which turns on the output. Therefore, using this technique, the output will be turned on sooner than desired.

How They Work

Synchronous M-Variable assignment statements were implemented as a solution to this problem. When one of these statements is encountered in the program, it is not executed immediately; rather, the action is put on a stack for execution at the start of the actual execution of the next move in the program. This makes the output action properly synchronous with the motion action.

In the modified program segment:

```
X10   ; Move X-axis to 10
M1==1 ; Turn on Output 1 synchronously
X20   ; Move X-axis to 20
```

The statement **M1==1** (the double-equals indicates synchronous assignment) is encountered at the beginning of the move to X10, but the action is actually not performed until the start of blending into the next move (X20).

Note:

With synchronous assignment, the actual assignment is performed where the blending to the new move begins, generally which is ahead of the programmed point. In LINEAR and CIRCLE mode moves, this blending occurs V*TA/2 distance ahead of the specified intermediate point, where V is the commanded velocity of the axis and TA is the acceleration (blending) time.

Also, notice that the assignment is synchronous with the commanded position, not necessarily the actual position. It is the responsibility of the servo loop to make the commanded and actual positions match closely.

In applications where PMAC is executing segmented moves (I13>0), the synchronous M-Variables are executed at the start of the first I13 spline segment after the start of blending into the programmed move.

Note:

Synchronous M-Variables after the last move or **DWELL** in the program do not execute when the program ends or temporarily stops. Use a **DWELL** as the last statement of the program to execute these statements.

Syntax

There are four forms of synchronous M-Variable assignment statements:

```
M{constant}=={expr}     ; Straight equals assignment
M{constant}&={expr}     ; AND-equals assignment
M{constant}|={expr}; OR-equals assignment
M{constant}^={expr}; XOR-equals assignment
```

In all of these forms, the expression on the right side of the statement is evaluated when the line is encountered in the program, ahead of the execution of the move. The value of the expression, the variable number, and the operator are placed on a stack for execution at the proper time.

Position Capture FLAGs

Interfacing to the FLAG inputs required for position capture was covered in detail in previous sections. Because these inputs are so closely associates with hardware limit switches, refer to the PMAC hardware and PMAC User Manual as well as any manuals supplied by the system integrator. Many PMAC based systems, for example X-Y tables, have already defined the operation of the limit switches and FLAG inputs. Hence, the use of these inputs must be coordinated with the system manufacturer's usage.

DAQ Signals

This manual cannot cover the wide selection of NI-DAQ boards or their signal sets. For the purposes of this section, a few general signals will be discussed that can be found in some form on most National Instruments DAQ boards and that can be used to trigger acquisitions and define the DAQ sample rates. Information that is more detailed can be found in the LabVIEW DAQ examples and tutorials and the manual for the DAQ board. All of the examples in this section use the standard LabVIEW examples.

Connections to the DAQ board are best done using one of National Instruments' many terminal blocks or breadboards. The terminals are well labeled and the chances for shorts are limited.

Analog I/O Channels

PMAC output signals may or may not be connected to the analog input channels on the DAQ board. In several of the examples that follow, the servo clock and JEQU signals are connected to Channel 0 and 1 for the purposes of demonstrating of how the clock looks. To configure, test, and operate these inputs, refer to the appropriate National Instruments manual.

Trigger and Scan Clock Connections

Acquisitions from PMAC can be triggered in many ways. Use the JEQU signal to start or stop acquisition at a precise position, or use general-purpose digital outputs and synchronous M-Variables. The servo clock and/or the JEQU signal can also be used for the scan clock, thereby synchronizing the DAQ sample rate with PMAC's primary timekeeper.

E Series Boards: Connect the start trigger to PFI0/TRIG1, the stop trigger to PFI1/TRIG2, and the scan clock to PFI7/STARTSCAN.

Legacy MIO Boards: Connect the start trigger to STARTRIG*, the stop trigger to STOPTRIG, and the scan clock to OUT2.

Note:

Two or more channels must be scanned when specifying an external scan clock.

Legacy MIO-16X, MIO-16F-5, and MIO-64F-5: The start and stop trigger pin is EXTTRIG*. Connect the scan clock to OUT2.

Note:

Two or more channels must be scanned when specifying an external scan clock.

Lab/1200 Series Boards: Connect the start or stop trigger to EXTTRIG. Hardware pre-triggering (start and stop) is not supported. Connect the scan clock to OUTB1.

For triggers and scan clocks on PC-LPM-16, DAQCard-500, and DAQCard-700, refer to the appropriate LabVIEW manual. To find the actual pin numbers, refer to the hardware user manual.

PmacDAQMove

There are dozens of approaches to configuring a particular PMAC/DAQ application. Consider placing status and position monitoring VIs inside the main DAQ polling loop. This requires organization of the configuration and maintenance of PMAC and DAQ device polling. For the purposes of this section, a multi-threaded model was selected consisting of a main VI to control PMAC and self contained DAQ VIs found in the LabVIEW examples. A few modifications were made to the DAQ examples such as defining defaults for the sample rates and channel configuration to demonstrate the sampling of the Compare-Equal output, Servo clock, and a simple analog signal. This allowed the examples to be created quickly and the operation of PMAC was validated in a more demanding execution environment.

PMAC and AT MI0-16 Signal Connections

The following examples were built using a PMAC Lite and a National Instruments' AT-MIO-16. The MIO card signals were accessed using a SC-2070 termination card. The various PMAC signals were accessed using various terminal blocks of the proper sizes. See the Hardware Reference Manual for the PMAC for a list of mating connectors.

In the examples, the DAQ triggers are driven by ENC1. There is no reason other encoders or combinations of signals from multiple motors cannot be used with simple modifications.

PMAC Signals

JRS232 (10-pin connector)	Pin 8 - Servo clock (SERVO) Pin 9 - Common
JEQU (10-pin connector)	Pin 1 - ENC1 Compare Equal Output (EQU1) Pin 10 - Common
JMACH1 (60-pin connector)	Pin 55 - ENC1 Home Flag Input (HMFL1) – Connected to External TTL Clock Pin 58 – Common

ATMIO Signals

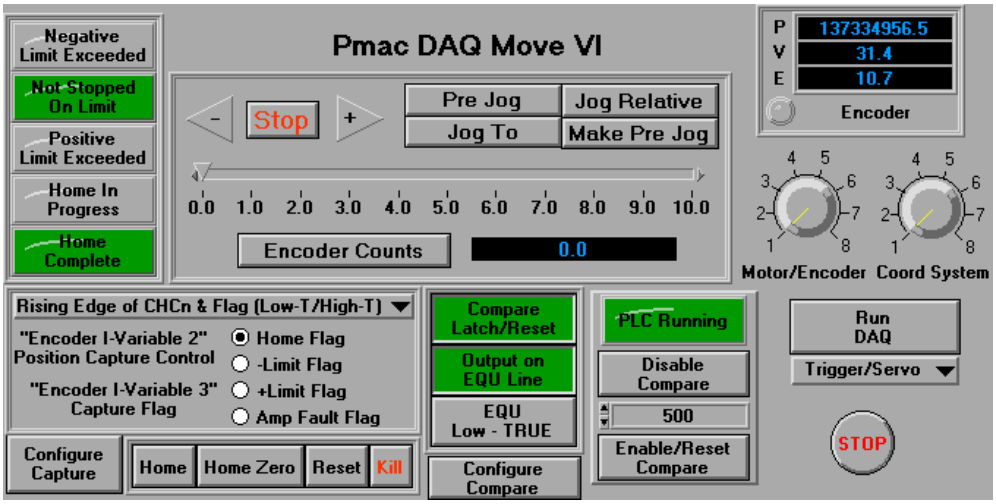
OUT2 – Scan clock – Wired to PMAC SERVO
EXT TRIG* - Start trigger – Wired to PMAC EQU1
CH0 – Wired to signal generator
CH1 – Wired to PMAC EQU1
CH2 – Wired to PMAC SERVO
Commons/Grounds – ATMIO DGND, ATMIO AGND, JEQU and JRS232 commons all wired together

The panel for PmacDAQMove is shown below. This VI is comprised of pieces from several of the previous examples. Use this VI to home motors, Jog them, configure capture and compare operations, and initiate three different DAQ operations.

If the system has been connected in the manner described above or something similar, begin testing the system by connecting EQU1 to an oscilloscope or by running the VI and using the DAQ board.

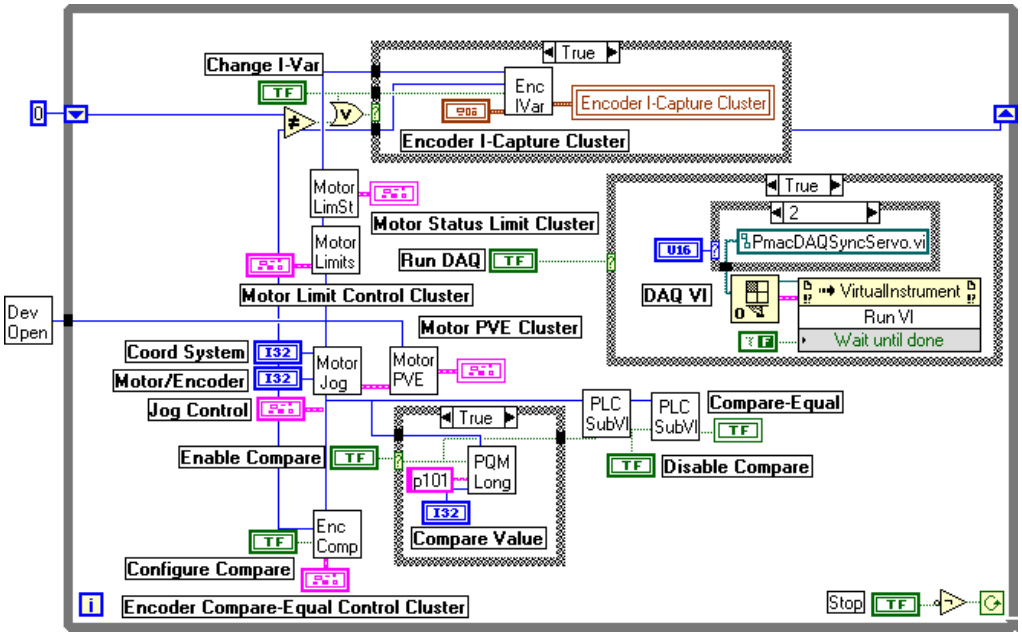
When the VI is started, select the capture flag configuration and click **Configure Capture**. Then home the motor. When this has been completed, set Compare configuration cluster booleans as shown below and click **Configure Compare**.

This VI uses the encapsulated PLC covered in the previous section to monitor and update the compare-equal operation. The value 500 in the numeric control specifies the pulse generation interval. This can be changed and then click **Enable/Reset Compare**. The green LED indicates that the PLC is running. When the motor is jogged using the jog controls, the PVE display update and a pulse train on the scope will display. Use level triggering on the scope. The pulse train will have stable intervals with jitter in the actual pulse widths. This is because a background PLC services the encoder. If a more stable pulse width is needed, change PLC 19 to PLC 0 in PmacPosCompGen.pmc and the reference to PLC 19 in PmacPosCompSetup.pmc to PLC 0. The foreground PLC will be serviced more regularly thereby resetting the output in a more deterministic manner.



If not using an oscilloscope, use the DAQ card to do the same thing. On the far right below the two selector knobs on the panel is a menu ring that allows the selection of three different DAQ VIs. Clicking the **Run DAQ** button starts the selected VI as a separate application thread. These VIs are slightly modified versions of standard LabView examples located in the Examples\Daq\Anlogin library supplied with LabVIEW.

The diagram uses the expected pieces from previous examples. The encapsulated PLC that generates the pulses is enabled by the setup PLC and can be disabled by the **Disable** button. The **Run DAQ** button uses the Server VI to start the selected PmacDAQ VI as a separate thread.



The three VIs, as named in \PmacDAQ, are described below. The name of the original VI in the LabVIEW example library is included in parentheses.

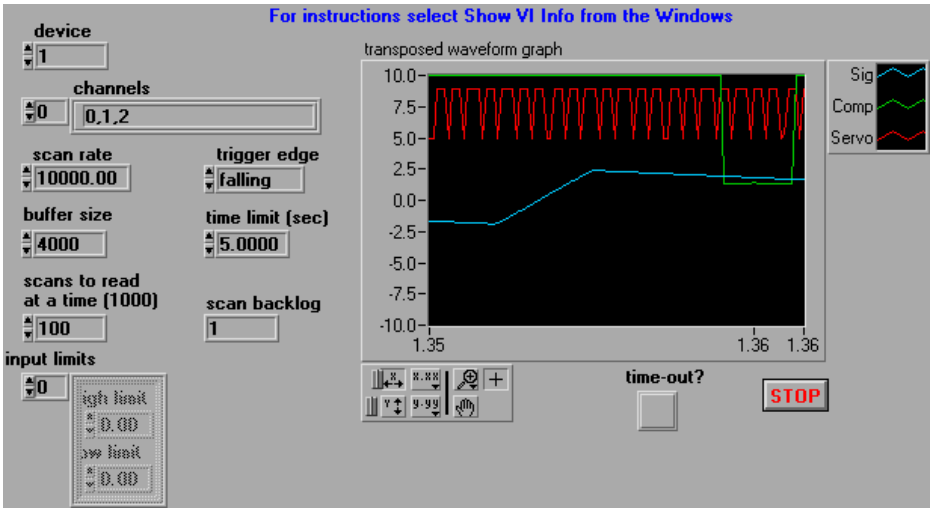
- PmacDAQTrigger – (Cont Acq&Graph (buffered) D-Trig.vi)
  This VI continuously acquires data from one or more analog input channels when a digital start trigger occurs. This is a timed acquisition, meaning that a hardware clock is used to control the acquisition rate for fast and accurate timing. It is a buffered acquisition, meaning that the data is stored in an intermediate memory buffer after it is acquired from the DAQ board. Data is retrieved from that buffer and displayed on the graph.

- PmacDAQSync – (Acquire N - Multi-Digital Trig.vi)
  This VI retrieves the specified amount of data from one or more analog input channels each time a digital start trigger, digital stop trigger, or digital start and stop trigger, occur. It shows how to trigger an acquisition multiple times while avoiding the overhead of configuration and buffer allocation each time. This is a timed acquisition, meaning that a hardware clock is used to control the acquisition rate for fast and accurate timing. It is a buffered acquisition, meaning that the data is stored in an intermediate memory buffer after it is acquired from the DAQ board.
- PmacDAQSyncServo – (Cont Acq&Graph ExtScanClk D-Trig.vi)
  This VI retrieves the specified amount of data from one or more analog input channels when a digital start trigger, digital stop trigger, or digital start and stop trigger, occur. This VI uses an external scan clock to continually retrieve data from one or more analog input channels. This VI will work only on devices where a scan clock signal can be connected externally. It is a buffered acquisition, meaning that the data is stored in an intermediate memory buffer after it is acquired from the DAQ board.

Single Trigger DAQ

PmacDAQTrigger is a LabVIEW example that waits for the external trigger supplied by EQU1 to begin asynchronous acquisition at the specified sample rate. In this version of the example, channels 0, 1, and 2 are sampled at 10KHz. The waveform chart in the panel below shows a triangle signal, the EQU trigger pulse, and the servo clock. Run this VI by selecting **Simple DAQ** in the menu ring, and clicking the **Run DAQ** button on PmacDAQMove after the PLC is configured and enabled and the motor is jogging. If the motor is not jogging, the VI will wait five seconds for the trigger and then time-out.

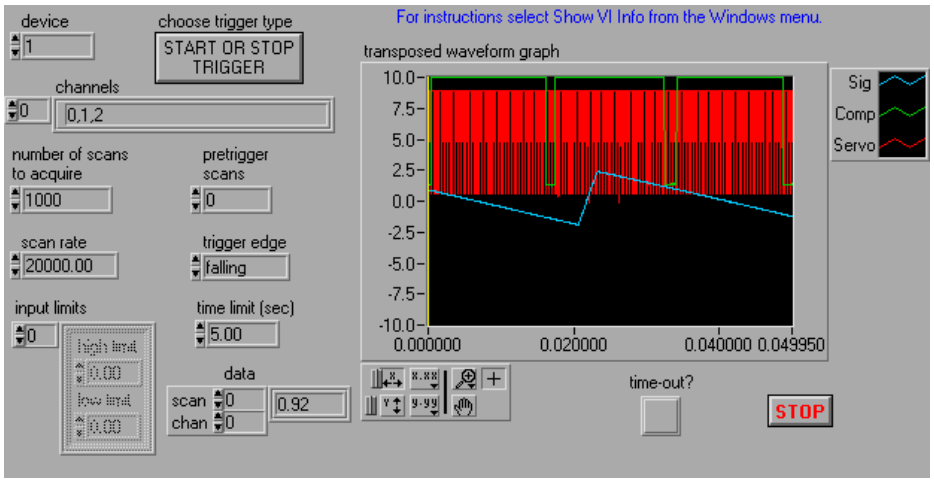
The default servo clock has a 442µS (2262 Hz) update rate. As can be seen in the chart the EQU signal is active low. Check the PmacDAQMove panel shown above to see that indeed the operation is configured for EQU Low – true. Hence, the initial trigger that started this acquisition started on the falling edge or leading edge of this pulse.



Multi-Trigger DAQ

PmacDAQSync is a LabVIEW example that performs repeated acquisitions synchronized by the external trigger supplied by EQU1 at the specified sample rate. In our default version of the example, channels 0, 1, and 2 are sampled at 20KHz. The waveform chart in the panel below shows a triangle signal, the EQU trigger pulse, and the servo clock. Run this VI by selecting **Trigger Only** in the menu ring, and clicking the **Run DAQ** button on PmacDAQMove after the PLC is configured and enabled and the motor is jogging. If the motor is not jogging, the VI will wait five seconds for the trigger and then time-out.

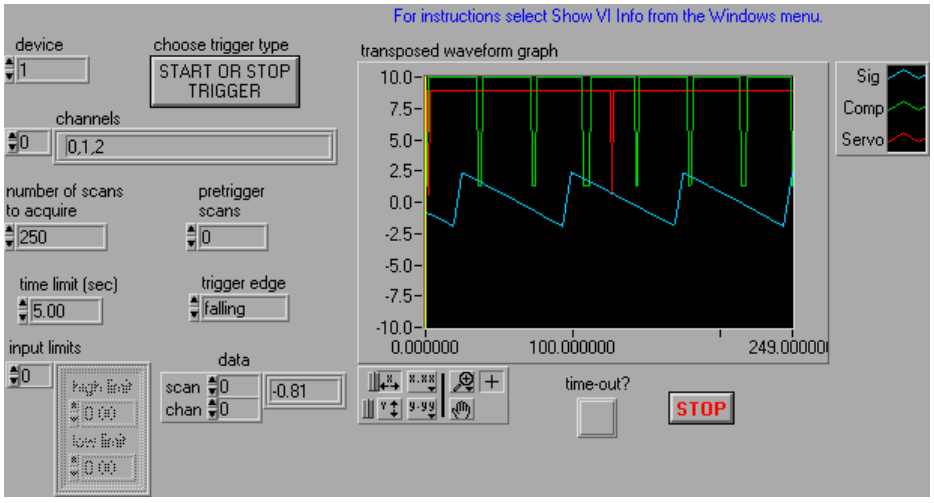
The EQU signal is configured to be active low. Hence, the first EQU pulse on the far-left starts with the falling edge of the signal. The default motor Jog rate configured by Ix22 is 32 counts/mS. With a position interval of 500 counts as configured by PmacDAQMove, this results in a pulse every 15.625 mS. Tthe next pulse occurs right around 15 mS in the chart.



Multi-Trigger DAQ with Servo Clock Sampling

PmacDAQSyncServo is a LabVIEW example that performs repeated acquisitions synchronized by the external trigger supplied by EQU1 using PMAC's servo clock as the scan clock. In the default version of the example, channels 0, 1, and 2 will be sampled at PMAC's default servo clock of 2262 Hz. The waveform chart in the panel below shows a triangle signal, the EQU trigger pulse, and the servo clock. Run this VI by selecting **Trigger/Servo** in the menu ring, and clicking the **Run DAQ** button on PmacDAQMove after the PLC is configured and enabled and the motor is jogging. If the motor is not jogging, the VI will wait five seconds for the trigger and then time-out.

The EQU signal is configured to be active low. The jitter in the EQU signal can be seen more clearly in this example. The analog signal is now sampled synchronously with the servo clock. Note that the servo clock trace in the chart is even caught once on CH2.



Further Sampling Options

The three examples presented here demonstrate that there are many options for triggering and controlling the sampling of the data.

- If the servo clock or the EQU signals are sampled along with the data, a time code is synchronized with the data.
- If a gather is performed on one of PMAC's position or velocity registers while using position compare intervals, the sampled data can be related with precise cycle-by-cycle motor positions or other motion characteristic.

Using these approaches, servo accurate positions can be achieved for every sample of the data. With LabVIEW's analysis tools, detailed data reductions can be performed that relate position, velocity, and physical measurements.

Other Interface Options

There is no reason that the DAQ boards DIO cannot be used to control PMAC's MIO inputs and vice versa. Multipurpose DAQ boards having timers can be used to generate position capture triggers, as can the board's D/A capabilities. Other possibilities include the use of the timers to generate time-base control for PMAC.

Although not covered here, use the sample PMAC generated clocks and signals to trigger and clock bench instruments used to communicate with using GPIB.

PCOMM 32 PRO CODE INTERFACE NODES

This section documents a basic framework for developing LabVIEW Code Interface Nodes (CINs) using Microsoft Visual C++. This topic is important if:

- More sophisticated control over a VIs implementation is needed.
- The **PmacDPR** collection of VIs for accessing dual-ported RAM must be modified.
- PMAC's interrupt capabilities will be used.

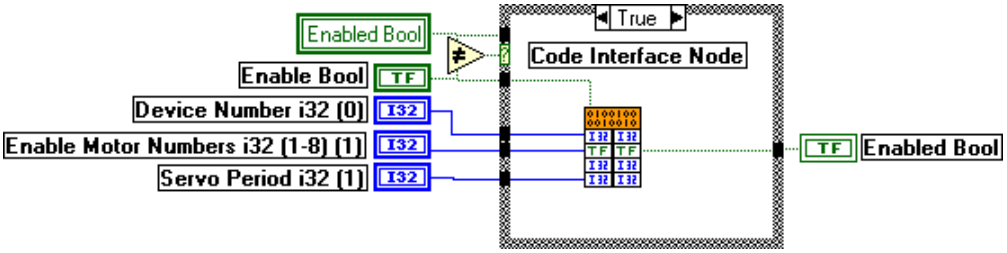
PMAC Panel Pro primarily interfaces to PComm 32 Pro using LabVIEW Call Library VIs to access specific PComm 32 Pro functions. In some instances, the need for increased speed, sophisticated manipulation of PMAC or the number of calls to PComm 32 Pro begins to create a nasty mess of Call Library VIs that becomes unmanageable.

In this section, a basic PmacCIN VI is introduced which is comprised of a VI, C/C++ source file, Microsoft Visual C++ workspace, and project file. The instructions show how to create the C file, modify the existing workspace and project file, compile the source file, and load the object file into the VI's Code Interface Node. LabVIEW 5.0 makes the process easy if some simple procedures are followed.

LabVIEW Code Interface Node Basics

What is a CIN?

Code Interface Nodes are VIs that call code written in C, directly from a block diagram. Many LabVIEW aficionados dislike CINs. However, there are instances where the logic required to implement an operation is much simpler to specify in C than LabVIEW's G. Furthermore, there are instances where the need for efficiency and speed suggest the use of CINs. Accessing PMAC DPR has both requirements. CINs can accept any LabVIEW data type including clusters and arrays as an input or output. The following illustration shows a CIN in a simple diagram.



LabVIEW provides several routines that make working with G data types easier. These routines support memory allocation, file manipulation, and data type conversion. Detailed documentation on these topics can be found in the LabVIEW Code Interface Reference Manual.

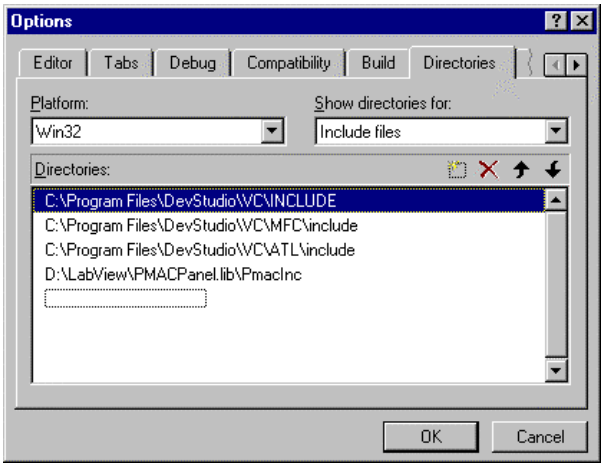
Setting up a PMAC Panel Pro CIN Configuration

There are two ways to create a project file for the CIN source code created by the LabVIEW. The information presented next is of general importance and leads to a much easier way to develop CINs for PMAC Panel Pro.

Adding PComm 32 Pro Include Path

To access PComm 32 Pro, add the following directories to the Visual C++ development environment by selecting **Tools»Options** to display a tab dialog. Click the **Directories** tab and select **Include fil**es in the Select directories drop down menu. Double click the outlined box in the directory and a list dialog box appears to select a directory for the include path. The dialog shown below already includes the D:LabView\PMAC Panel Pro.lib\PmacInc directory.

If PComm 32 Pro is installed, use this directory. The result should look something like the following:



Adding Pmac.lib to Project

To link the CIN to produce the lsb code resource, include Pmac.lib located in PMAC Panel Pro.lib\ PmacInc or the PComm 32 Pro installation directory. Select **Project>>Add To Project>>Files** and **Pmac.lib** in the locations noted. Also add the file and the path to the Library files selection in the Tool options the same way that the include path was added.

Configuring the IDE

Detailed instructions on the steps required to configure a project so that it will successfully compile a C file into a loadable code resource for the CIN are available. This is a bothersome process if it is done a number of times. It may be necessary to do this the first time a CIN is created and compiled. After that, use the techniques detailed next to duplicate the project file.

The Easy Way to Add New Projects

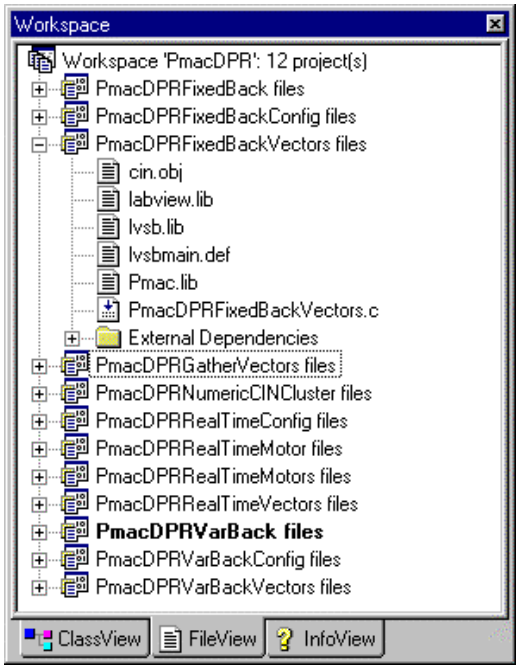
The easy way to create new CIN projects is to create a copy of the PmacCIN.dsp project file and workspace PmacCIN.dsw located in \PmacCIN and modify them using notepad or Microsoft Word. Microsoft does not recommend this, but it saves a lot of configuring when creating a new CIN.

The project file has the keyword PmacCINBase used 21 times in it. The workspace has the keyword PmacCINBase used once. Copy the files to a new directory, give them new names such as PmacMyCustomCIN.dsp and PmacMyCustomCIN.dsw, and replace all references to PmacCINBase with PmacMyCustomCIN. Then open the new workspace in Visual C++ and compile the project using the LabVIEW created C source file named PmacMyCustomCIN.c.

If the installation is different from that contained in PmacCIN**,** try to edit the dsp or dsw files. If there are too many changes, create a Microsoft Visual C++ environment from scratch, add the paths and libraries for PMAC, and get one project to work. From this point on, copy the dsp, edit it, and insert it into an existing workspace. It takes a little work the first time, but adding new CINs is easy after that.

Multiple CIN Projects in a Workspace

Managing multiple CIN projects gets troublesome quickly if each CIN has its own project and workspace. Navigate to the PmacDPR directory and open the Microsoft Visual C++ workspace named PmacDPR.dsw and note that the workspace, shown below, has 12 projects in it. PmacDPRVarBack is the currently active project and will be compiled when the Build command is selected. This figure is also instructive in that the project PmacDPRFixedBackVectors is open. It shows that any LabVIEW CIN project requires cin.obj, labview.lib, lvsb.lib, and lvsbmain.def. For PMAC Panel Pro Pmac.lib is also required. The file PmacDPRFixedBackVectors.c is the C source file created by LabVIEW for the CIN node and contains the actual code to accomplish the desired task.



Creating a CIN C-Stub for PComm 32 Pro

After placing the CIN VI in the diagram, wiring the inputs, and creating the C-source file to edit the source. The following code was created for the Code Interface Node in the VI PmacCIN.

CIN Source File

```
#include "extcode.h"
CIN MgErr CINRun(int32 *Device_Number_i32_0_,
LVBoolean *Enable_Bool,
int32 *Enable_Motor_Numbers_i32_1_8_1_,
int32 *Servo_Period_i32_1_);
CIN MgErr CINRun(int32 *Device_Number_i32_0_,
LVBoolean  *Enable_Bool,
int32 *Enable_Motor_Numbers_i32_1_8_1_,
int32 *Servo_Period_i32_1_) {
/* ENTER YOUR CODE HERE */
return noErr;
}
```

To access PComm 32 Pro capabilities, add the line `#include <pmacu.h>.`

Then all of PComm 32 Pro's capabilities can be utilized.

DPR – DUAL-PORTED RAM

Every collection of VIs presented so far uses ASCII command strings to communicate between PMAC and the host. This is independent of whether the actual transfer between the host and PMAC takes place over a serial port, the bus, or DPR. The parsing, formatting, handling, and interpretation of the commands is responsible for most of the time required for communication – even communication that takes place using dual-ported RAM (DPR).

Dual-ported RAM provides seven other mechanisms for the transfer of limited and specific sets of numeric data between the host and PMAC that requires far less handling. This results in much faster transfers that may be advantageous in an application.

These mechanisms are:

- **Fixed Real Time Data Buffer** – Automatic copying of limited servo data to the host at a specified servo rate
- **Fixed background Data Buffer** – Automatic copying of limited motion program data to the host on an as-requested basis
- **Variable Background Data Buffer** – Automatic copying of user specified data to the host
- **General Numeric Access** – Bi-directional transfer of numeric data between the host and PMAC using any DPR addresses not dedicated to another operation
- **Control Panel** – Emulation of PMAC's hardware control panel
- **Binary Rotary Buffer** – Execution of motion programs loaded by the host on an as-requested basis
- **Real Time Data Gathering** – Automatic copying of user specified data to the host

Of these seven mechanisms, one through four are completely supported by the PmacDPR collection of VIs. Mechanism 5 is not required in that the same capabilities, albeit slightly slower, are provided using the existing VIs. Mechanism 6 is way beyond the requirements of anything a developer should attempt with PMAC Panel Pro. Generally, PMAC does not use mechanism 7.

In this section, four collections of VIs are introduced in PmacDPR that provide the capabilities required for mechanisms 1-4. These are:

- **PmacDPRFixedBack** – Fixed Background data buffer
- **PmacDPRNumeric** – General numeric access to unallocated DPR memory
- **PmacDPRRealTime** – Fixed Real Time data buffer
- **PmacDPRVarBack** – Variable Background data buffer

In each of the four collections contains:

- A configuration VI to enable and configure the operation
- VIs to read or write the data using convenient clusters and types
- VIs to buffer multiple data samples into vectors for charting and analysis.
- Examples of all capabilities

Several indicators and controls are provided to handle the data in easy to use clusters. In many instances, previously introduced concepts and clusters are used in ways that tightly integrate the new capabilities available through DPR into PMAC Panel Pro's familiar architecture.

Once the structure of a configuration, fetch, and vector VI are understood for one of the collections the others will be readily duplicated. Some applications require a slightly different architecture than that presented here. If so, modify the existing CINs to suit any particular needs.

Required Background Understanding

Before reading this material, there must be some familiarity with PMAC's DPR capabilities. This can be found in the PMAC User Manual, PMAC Software Reference Manual, and PMAC Dual-Ported RAM manual. The architecture of the CINs to support the VIs is influenced by the PComm 32 Pro DPR API so having that manual available is also necessary. In particular, the section on PComm 32 Pro DPR Features should be read before proceeding.

General Architecture Notes

PMAC Panel Pro's DPR support is designed to be simple and extensible. There is not a lot of error checking. The basic assumption is that there is a DPR card with the PMAC. PMAC DPR does not check automatically for this nor does it enable or disable itself automatically. As the examples are inspected, note that each mechanism has a configuration VI. As applications are developed, move these into the PmacDevOpen VI so that opening PMAC also enables DPR.

PmacDPRRealTime

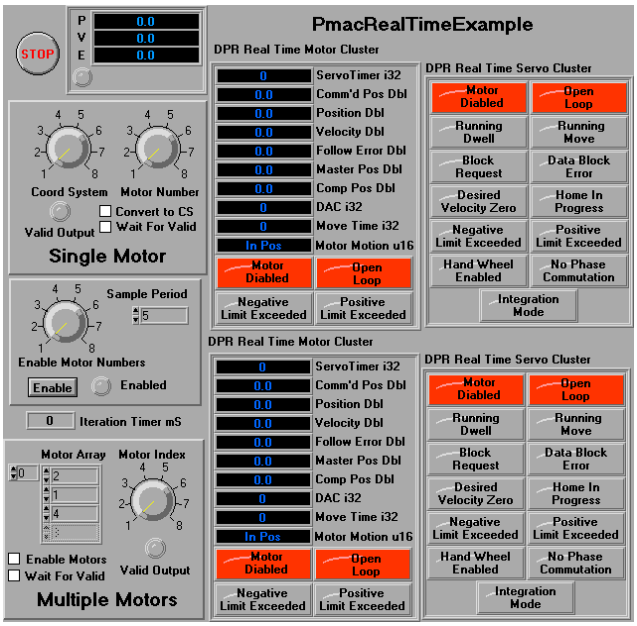
PMAC's Real Time data buffer mechanism automatically copies 27 selected Motor Calculation Registers from their native PMAC locations to DPR locations at a specified servo-cycle sample rate. PComm 32 Pro supplies a set of routines to read these values from DPR and convert them into legitimate Intel formats. This process requires some handshaking between PMAC and the host to avoid collisions when accessing DPR from the host.

To minimize the work and simplify the interface, the PmacDPRRealTime collection bundles the 23 most useful items them into clusters. This ensures that all data items are gathered during the same servo cycle. It also prevents having to wire 27 VIs.

PmacDPRRealTimeExample

The following example demonstrates three PmacDPRRealTime VIs. One VI configures and enables the operation of Real Time data buffering and two fetch the data. PmacDPRRealTimeMotor collects the data for a single motor. PmacDPRRealTimeMotors collects the data for a set of motors. Grouping the fetch of data for multiple motors into a single VI ensures that the data for each motor will be from the same servo cycle.

The panel for the example is shown below. The panel demonstrates the operation for a single motor on the top and multiple motors on the bottom. On the right are two clusters for displaying the data fetched from DPR. On the right are controls for selecting which motor or motors and controlling the fetch from DPR. On the left in between the two is a small block of controls to enable DPR Real Time data buffering.



To execute the example, how many motors PMAC should copy to DPR using the **Enable Motor Numbers** knob. Sample Period is the number of servo cycles between copies to DPR. The default value of five indicates that PMAC will update the Real Time data buffer every five servo cycles. With a default servo rate of 2.2 kHz, this corresponds to a 400 Hz sample rate.

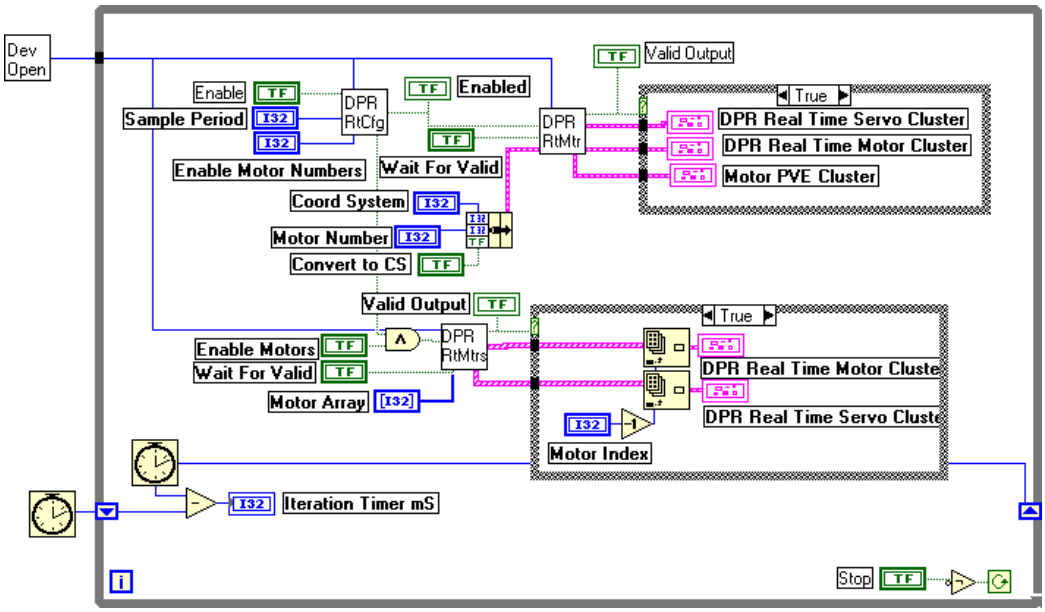
Click **Enable** to start PMAC's Real Time data buffer. Updates should take place immediately in the DPR Real Time Motor Cluster and DPR Real Time Servo Cluster on top. Most noticeably, the Servo Timer will increment rapidly, reflecting the servo time the sample was taken. If four motors are enabled, use the Motor Number knob in the Single Motor box to fetch and display the data for the corresponding motor.

Note the familiar PmacMotorPVE cluster on the top left displays the PVE as a subset of the data contained in the DPR Real Time Motor Cluster. Check the **Convert to CS** box and the same conversion to coordinate system units covered in several earlier sections are applied to the PVE data gathered from DPR.

Below this collection of controls is an indicator labeled Iteration Timer that displays the time in mS for each loop iteration. On average, the fetch and display update of DPR Real Time data for a single motor takes 1-2mS.

Notice the green LED in the Single Motor box flickering on and off. When PMAC copies data to DPR, it sets a Busy Bit indicating that it is accessing DPR. During this time, the host, running this VI, cannot access DPR. To avoid possible problems, the VI simply indicates that it did not perform a successful fetch. If the **Wait For Valid** box is checked, the VI will continue placing calls to the associated CIN until it performs a successful read.

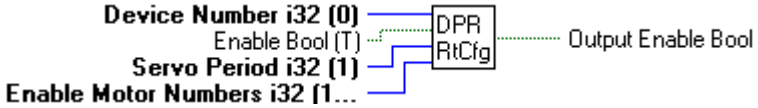
The demonstration on the bottom of the panel is for multiple motors. To fetch the data for multiple motors, supply an array of motor numbers and enable the fetch. The VI then fetches the data for the specified motors and returns an array of clusters. The time required for this multiple motor fetch is on the order of 1-2 mS. In this example, select a Motor Index for display. As shown, a Motor Index of 1 displays the data for Motor 2.



The diagram for the example requires three VIs. One VI configures the operation, one fetches the data for a single motor and one fetches the data for multiple motors. **CASE** statements are used to control the update of the display clusters. The architecture of the PmacDPR VIs is different from most of those introduced already. Whereas almost all other collections operate in a query response mode that is always enabled, these VIs require the enabling of specific capabilities by a configuration VI. Hence, most PmacDPR VIs have an enable input that prevents them from querying DPR until it is enabled.

On the top left the PmacDPRRealTimeConfig VI requires a Sample Period, Enable Boolean, and an integer indicating how many motors to copy. The enable input is not latched. When it is true, the Real Time data buffer is enabled. When it is false, the operation is disabled.

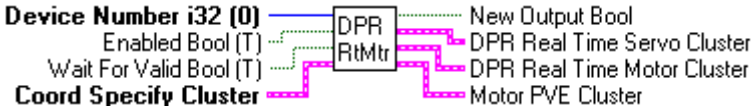
- PmacDPRRealTimeConfig – DPR Real Time Fixed buffer is configured to update motor information every Servo Period for all motor between 1 and Enable Motor Numbers when Enable is true. Operation is disabled when Enable is false. Output Enable is true when operation is enabled. The VI maintains the state. Operation of DPR Real Time buffers overlaps with DPR Fixed Background operation in that the number of motors enabled must be the same.



PmacDPRRealTimeMotor fetches DPR data for a single motor. This has an optional enable signal, in this case provided by the configuration VI. It also has an optional Wait for Valid input and a Coord Specify Cluster that is used to specify the Motor Number and standard Coordinate System conversions for the production of the PVE cluster.

- PmacDPRRealTimeMotor – Query PMAC DPR for the Real Time Fixed buffer Motor and Servo data. When Enabled is true (the Default state), the data for Motor Number is fetched and used to build DPR Real Time Motor Cluster and DPR Real Time Servo Cluster. Motor PVE Cluster contains data in encoder counts or coordinate system units depending on the state of Coord Specify Cluster. See PmacMotorPVE for details on how this is done.

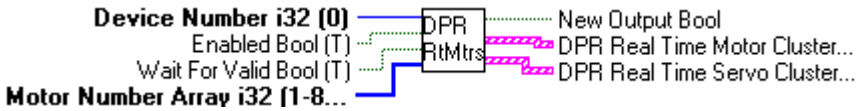
A successful query of PMAC's DPR depends on whether PMAC is accessing the memory. If Wait for Valid is true, the VI places queries to PMAC until a successful read at which time New Output is true indicating valid output data. If Wait for Valid is false, the query may or may not succeed. If the query fails, New Output is false and the output clusters contain the data fetched during the last read.



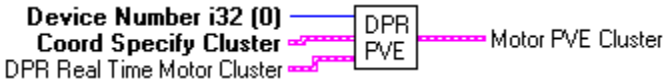
The fetching of multiple motor data by PmacDPRRealTimeMotors requires an array of motors and produces an array of clusters. This VI differs from PmacDPRRealTimeMotor in that it does not provide Coordinate System conversions. This is not provided because it would require assembling an array of Coord Specify Clusters. If the Coordinate System conversion of Real Time data for multiple motors is needed, use the PmacDPRRealTimePVE and apply the transformation to the individual cluster elements in the output arrays.

- PmacDPRRealTimeMotors – Query PMAC DPR Real Time Fixed buffer Motor and Servo data for the list of motors specified in Motor Number Array. When Enabled is true (the Default state), the data for the specified motors is fetched and used to build an array of DPR Real Time Motor Clusters and DPR Real Time Servo Clusters.

A successful query of PMAC's DPR depends on whether PMAC is accessing the memory. If Wait for Valid is true, the VI places queries to PMAC until a successful read at which time New Output is true indicating valid output data. If Wait for Valid is false, the query may or may not succeed. If the query fails, New Output is false and the output clusters contain the data fetched during the last read.

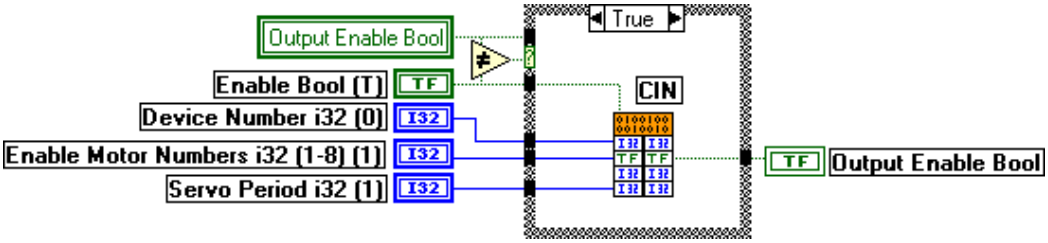


- PmacDPRRealTimePVE – Extract position, velocity, and following error from DPR Real Time Motor Cluster assuming Motor Number operating in Coord Number. Assemble the measurements into Motor PVE Cluster. If Convert is true, convert the measurements to CS units. Otherwise, leave them in encoder counts.



PmacDPRRealTimeConfig CIN

The simple diagram for PmacDPRRealTimeConfig is shown below. Note that the Device Number, Enable Motor Numbers, and Servo Period are all passed to the CIN. Even the Enable is passed. The CIN returns an Output Enable signal that indicates whether the configuration and enable operation succeeded. The only unique characteristic of this VI is the compare operation between the Output Enable and Input Enable. When these values are not equal, the true case executes enabling or disabling the operation as defined by the input Enable.



Many things can be done with a CIN. Have a copy of the LabVIEW CIN Reference Manual when working with these until familiar with how LabVIEW passes parameters. This is true especially for clusters and arrays. The actual C code that implements this CIN is shown below:

CIN Source File

```
#include "extcode.h"
#include <pmacu.h>
#include <dprrealt.h>
```

Stubs for Advanced CIN Functions

```
UseDefaultCINInit
UseDefaultCINDispose
UseDefaultCINAbort
//UseDefaultCINLoad
UseDefaultCINUnload
UseDefaultCINSave
// -- This a GLOBAL variable.
BOOLEAN Enabled = FALSE;
// --
CIN MgErr CINRun(int32 *Device_Number_i32_0_,
LVBoolean *Enable_Bool_T_,
int32 *Enable_Motor_Numbers_i32_1_8_1_,
int32 *Servo_Period_i32_1_);
CIN MgErr CINRun(int32 *Device_Number_i32_0_,
LVBoolean *Enable_Bool_T_,
int32 *Enable_Motor_Numbers_i32_1_8_1_,
int32 *Servo_Period_i32_1_) {
      /* -- When not currently enabled and Enable_Bool_T_ == LVTRUE
      enable the Fixed buffer for the specified number of motors -- */
      if (!Enabled && *Enable_Bool_T_ == LVTRUE) {
PmacDPRSetMotors(*Device_Number_i32_0_,
*Enable_Motor_Numbers_i32_1_8_1_);
            PmacDPRRealTime(*Device_Number_i32_0_,
*Servo_Period_i32_1_,
1);
            Enabled = TRUE;
      }
      /* -- When currently enabled and Enable_Bool_T_ == LVFALSE
      disable ALL background operations. -- */
      else if (Enabled && *Enable_Bool_T_ == LVFALSE) {
```

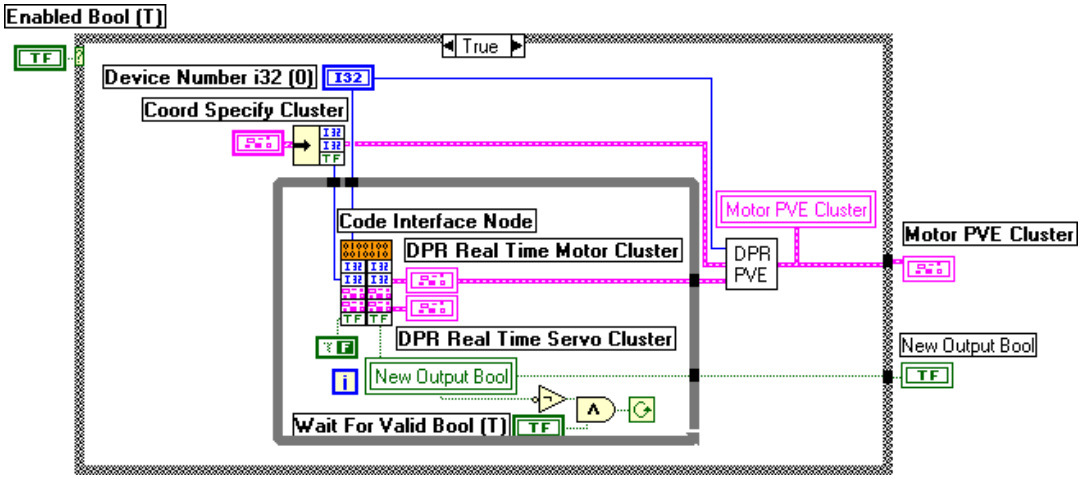
```
            PmacDPRRealTime(*Device_Number_i32_0_,
*Servo_Period_i32_1_,
0);
            Enabled = FALSE;
      }
      *Enable_Bool_T_ = Enabled;
      return noErr;
}
// When first loaded make sure Enable flag is FALSE
CIN MgErr CINLoad(RsrcFile rf)
{
      Enabled = FALSE;          // Indicate DPR Fixed Real Time disabled

      return noErr;
}
```

This particular CIN has two functions CINLoad and CINRun. LabVIEW creates the function and data type declarations such as clusters and arrays required by CINRun. The function CINRun is called when the VI containing the CIN is executed. CINLoad is executed when the VI is first loaded.

In this sample, two functions from PComm 32 Pro are used. PmacDPRRealTime enables and disables the DPR Real Time data buffer and PmacDPRSetMotors sets the number of motors to copy. The logic of the **IF** statement uses the current enable state of the operation and the desired state passed in by *Enable_Bool_T_ to turn the DPR Real Time data buffer on or off. Notice the type of the Boolean and that most parameters are passed as pointers to data.

PmacDPRRealTimeMotor CIN

The diagram for PmacDPRRealTimeMotor is shown below. Note that the Device Number, Motor Number, and a false Boolean constant are passed to the CIN. The CIN returns two clusters and a Boolean indicating whether the fetch operation succeeded. The CIN is only executed when the input Enabled is true. If Wait for Valid is true, the while loop will execute the CIN until New Output is true thereby waiting until the fetch from DPR succeeds. Repeated requests for DPR data is done by the diagram rather than the CIN code. If the C-code waited in a while loop that was never satisfied, the application could not be aborted.



This VI's CIN code has a simple structure but places many calls to PComm 32 Pro. This is the reason this VI was selected to implement with a CIN rather than 20 or 30 Call Library Function VIs. To simplify the code, several lines from the actual source have been removed to demonstrate the basic ideas.

CIN Source File

```
#include "extcode.h"
#include <pmacu.h>
#include <dprrealt.h>
```

Types Defined by LabVIEW When the Stub was Created

```
typedef struct {
      int32 ServoTimer_i32;
      float64 Comm_d_Pos_Dbl;
      float64 Position_Dbl;
      float64 Velocity_Dbl;
      float64 Follow_Error_Dbl;
      float64 Master_Pos_Dbl;
      float64 Comp_Pos_Dbl;
      int32 DAC_i32;
      int32 Move_Time_i32;
      uInt16 Motor_Motion_u16;
      LVBoolean Motor_Activated;
      LVBoolean Open_Loop;
      LVBoolean Neg_Limit_Exceeded;
      LVBoolean Pos_Limit_Exceeded;
} TD1;
typedef struct {
      LVBoolean Home_In_Progress;
      LVBoolean Block_Request;
      LVBoolean Desired_Velocity_Zero;
      LVBoolean Data_Block_Error;
      LVBoolean Dwell_In_Progress;
      LVBoolean Integration_Mode;
      LVBoolean Running_Move;
      LVBoolean Open_Loop;
      LVBoolean Phased_Motor;
      LVBoolean Hand_Wheel_Enabled;
      LVBoolean Neg_Limit_Exceeded;
      LVBoolean Pos_Limit_Exceeded;
      LVBoolean Motor_Activated;
} TD2;
// --
CIN MgErr CINRun(int32 *Device_Number_i32_0_, int32 *Motor_Number_i32_1_8_1_,
TD1 *DPR_Real_Time_Motor_Cluster,
TD2 *DPR_Real_Time_Servo_Cluster
LVBoolean *ValidData){
      int32 DevNum = *Device_Number_i32_0_;  // Shorter dereferenced name
      int32 MNum = *Motor_Number_i32_1_8_1_ - 1;
      SERVOSTATUS ServoStatus;
      // --  Tell PMAC we're doing our thing
      PmacDPRSetHostBusyBit(DevNum, 1);
      // -- Check if PMAC is busy doing its thing
      if (!PmacDPRGetPmacBusyBit(DevNum)) {
            // -- Fetch all of the available data
            DPR_Real_Time_Motor_Cluster->ServoTimer_i32 =
PmacDPRGetServoTimer(DevNum);
            …
            DPR_Real_Time_Motor_Cluster->Motor_Activated =
PmacDPRMotorEnabled(DevNum, MNum) == 0 ? LVFALSE : LVTRUE;
```

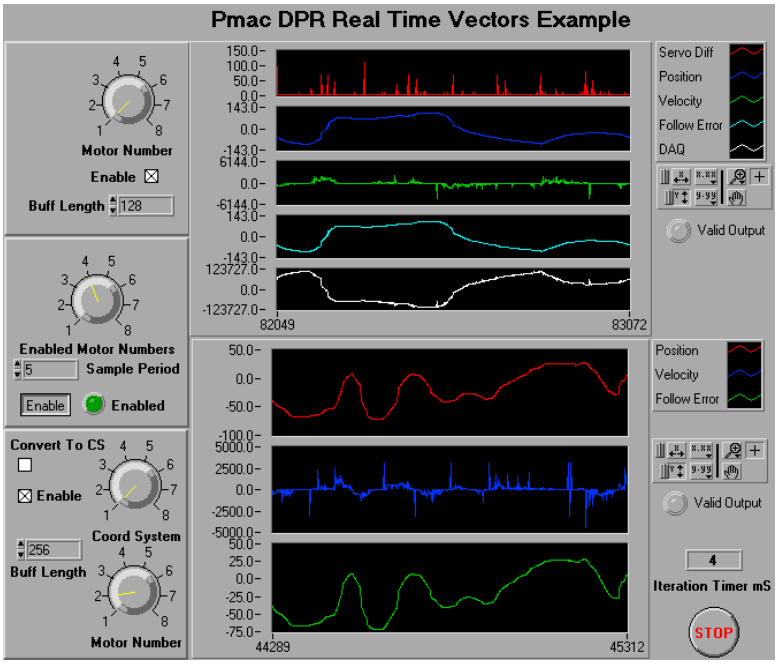
```
            // -- ServoStatus – Fetch cluster and then individual items
            ServoStatus = PmacDPRMotorServoStatus(DevNum, MNum);
            DPR_Real_Time_Servo_Cluster->Home_In_Progress =
ServoStatus.home_search == 0 ? LVFALSE : LVTRUE;
            …
            // --

            *ValidData = LVTRUE;       // New data for caller
      }
      else {
            *ValidData = LVFALSE;      // Sorry - no new data
      }
      PmacDPRSetHostBusyBit(DevNum, 0);// PMAC can do its thing
      return noErr;
}
```

Check the PComm 32 Pro Reference Manual and see that the checking and setting of the DPR Busy Bit is required by PMAC. When the unfilled CIN node stub was created LabVIEW generously declared the CINRun parameter order, names, and data types. CINRun's job is to fetch DPR data from PMAC using the PComm 32 Pro functions and fill the LabVIEW data types passed by the caller with the data. It is actually simple. The parameters DPR_Real_Time_Servo_Cluster and DPR_Real_Time_Motor_Cluster are pointers to the data types provided by LabVIEW. Calls are placed to PComm 32 Pro and data of the proper type is assigned to the members of the data types. There is one catch here. Note that Booleans returned by PComm 32 Pro are converted to LVTRUE and LVFALSE before being assigned to the members of the clusters. This precaution avoids possible mismatches in data types.

PmacDPRRealTimeVectorExample

The following example demonstrates a very powerful PmacDPR technique that takes multiple samples over time. This is accomplished by placing repeated CIN that build vectors for the desired items. This creates a simpler and faster VI diagram because building the vector is completed by the CIN and the data is returned by the CIN only when a vector of a specified length is built.

The panel for the example is shown below. The panel demonstrates the fetching of vectors for the purposes of driving a real-time chart. On the top left is a knob for selecting a motor and specifying the number of samples to accumulate before updating the chart. On the bottom, the same operations are performed but motor items such as Position, Velocity, and Following Error are converted to CS units in the standard manner defined for PmacMotors. On the left between the two sets of controls is a small block of controls to enable DPR Real Time data buffering.



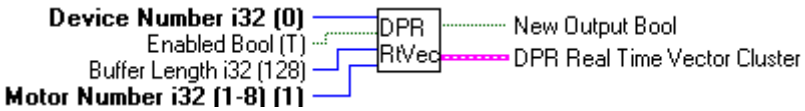
To execute the example, select how many motors PMAC should copy to DPR using the **Enable Motor Numbers** knob. Sample Period defines the servo-sampling interval. Then click **Enable** to start the DPR Real Time data buffer. As with PmacDPRRealTimeExamp**,** a motor can be selected to fetch, and then check the **Enable** box for the top or bottom chart.

The PmacDPRRealTimeVectors VI that actually processes the request for a fetch does a little more book keeping and buffers the data in arrays prior to passing it back to the caller. It returns a PmacDPRRealTimeVectors cluster from which the desired items can be selected and plotted as shown here. This cluster differs from PmacDPRRealTimeMotor and PmacDPRRealTimeServo. Generally many of the items in the clusters would not be of interest in a time-vector. To use these, modify PmacDPRRealTimeVectors to support them.

The demonstration on the bottom panel is different from that on top in that it has a longer buffer (256 samples vs. 128 samples) and a CS transformation can be applied.

- PmacDPRRealTimeVectors - Query PMAC DPR for the Real Time Fixed buffer Motor and Servo data. When Enabled is true (the Default state), the data for Motor Number is fetched and used to build DPR Real Time Motor Cluster and DPR Real Time Servo Cluster. Motor PVE Cluster contains data in encoder counts or coordinate system units depending on the state of Coord Specify Cluster. See PmacMotorPVE for details on how this is done.

A successful query of PMAC's DPR depends on whether PMAC is accessing the memory. If Wait for Valid is true, the VI places queries to PMAC until a successful read at which time New Output is true indicating valid output data. If Wait for Valid is false, the query may or may not succeed. If the query fails, New Output is false and the output clusters contain the data fetched during the last read.



If both portions of the example are enabled, ensure that each requests the data for a different motor.

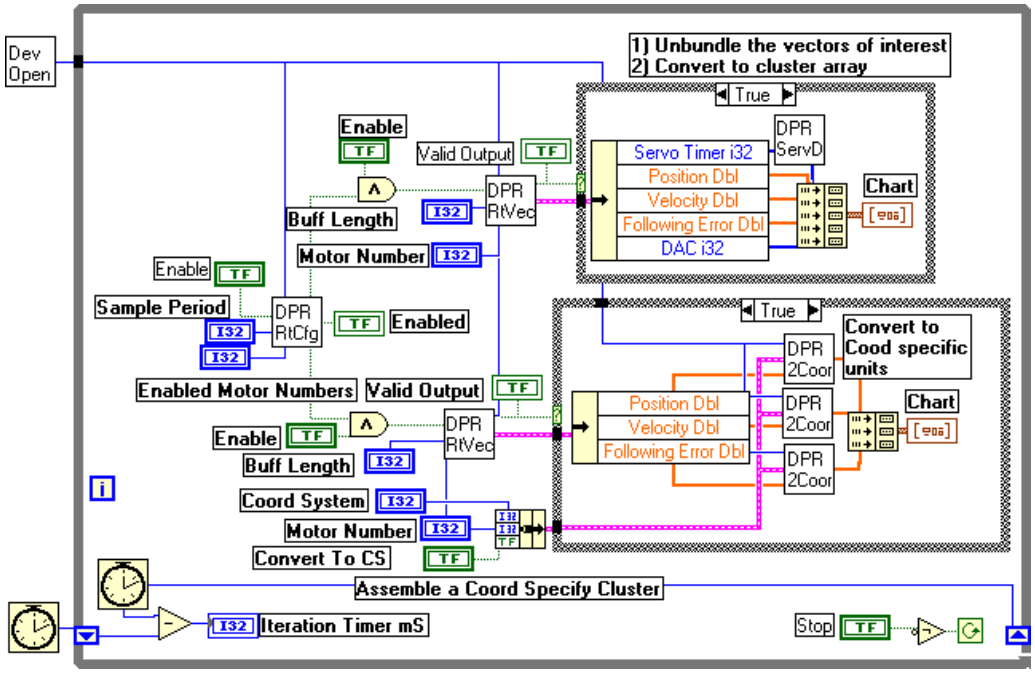
Servo Accurate Sampling

There is an important issue regarding the PmacDPR Vector VIs that must be understood to avoid misconceptions. PMAC is a very fast real-time controller that generates more data than could possibly be used in any given application. Previously, the PmacGather collection of ICVs that utilized PMAC's data gathering capabilities was discussed. The gather facility gathers and buffers a specified set of items at a specified servo rate using PMAC memory. The gathered data can be transferred to the host later for decoding and use. The PmacDPR Vector VI collects the data in host memory on an as-it-gets-there basis. Samples will be missed when the application is busy with other operations. This can be seen in the example's red chart strip of the Servo Timer difference. This strip chart is the difference between successive Servo Timer samples and reflects the jitter in the sampling. Most samples are taken every five servo cycles in this example. When a complete buffer is accumulated, passed back to the VI, and then used to update the chart, the sampling interval experiences a blip of approximately 75 mS. 5 and 10 mS blips in between the major buffer updates can be noticed.

From a practical point of view, the Servo Timer vector definitively identifies the time each sample was taken. This can be used to resample the other data vectors, or handled however chosen. Until Delta Tau includes DPR data gathering in PComm 32 Pro, use regular data gathering if servo accurate sampling is required. This form of gathering does not support strip charting on a continuous basis.

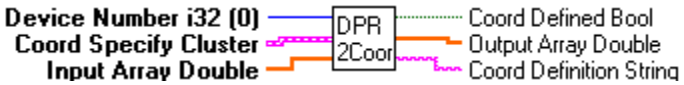
As can be seen in this panel, the interval required to process both fetches is between two and four mS without the update of the charts. When the vectors have been accumulated, the updating of the charts requires between 75 and 100 mS.

The diagram for the example is similar to that of PmacDPRRealTimeExamp. The configuration VI is the same. The Vector VI PmacDPRRealTimeVectors is similar to PmacDPRRealTimeMotors in that it takes an Enable input. It also has a Buffer Length input. Rather than a Coord Specify Cluster, provide a Motor Number. The decision to do this is based on the view that whereas single Servo sample data for a motor might be used to drive a PVE type of panel cluster. This is not true for a Vector operation. The update of the chart is wrapped in a case structure. Do not use the cluster of arrays unless Valid Output is true. The arrays should have zero length, but this may cause problems.



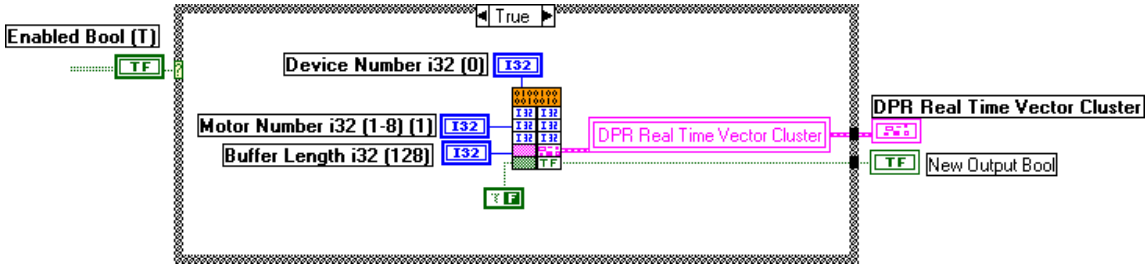
The vector fetch in the bottom half of the diagram unbundles the returned cluster of vectors and performs a CS conversion on the elements of the selected vectors using PmacDPRMotorVecToCoord. This is a vector version of the standard PmacCoord VIs.

- PmacDPRMotorVecToCoord - Coord Specify Cluster specifies a motor within a CS and an attempt to convert Input Array from encoder counts to CS units. If the motor is not defined in the CS, no conversion is applied. If the motor is defined and Convert is true, Coord Defined is true, and Output Array is scaled from encoder counts to CS units. Coord Definition is a string specifying Output Value units as Encoder or the CS definition of the motor.



PmacDPRRealTimeVectors CIN

The diagram for PmacDPRRealTimeVectors is shown below. Note that the Device Number, Motor Number, and Buffer Length are all passed to the CIN when Enable is true. The CIN fetches the DPR samples and builds the vectors on every execution of the CIN. 99% of the time the CIN returns a false value for New Output because it still has more samples to accumulate. When it has accumulated Buffer Length of samples, it copies them to DPR Real Time Vector Cluster and returns a true for New Output.



The CIN code for this type of VI is a bit more complex than those presented already. It requires a data buffer for storing the accumulated samples and it requires some manipulation of the vectors in the returned cluster. As defined, the data buffers are global thereby making them accessible to any reentrant copy of the VI. The topic of handling arrays in CINs is thoroughly covered in the LabVIEW CIN Reference Manual. To simplify the code, several lines have been removed from the source to demonstrate the basic ideas.

CIN Source File

Standard Includes and Definitions – See Actual Source File

```
// -- i32 vector
typedef struct {
      int32 dimSize;
      int32 Value[1];
} TD2;
typedef TD2 **TD2Hdl;
// -- float64 vector
typedef struct {
      int32 dimSize;
      float64 Value[1];
} TD3;
typedef TD3 **TD3Hdl;
// -- Cluster of arrays
typedef struct {
      TD2Hdl Servo_Timer_i32;
      TD3Hdl Position_Dbl;
      TD3Hdl Velocity_Dbl;
      TD3Hdl Commanded_Pos_Dbl;
      TD3Hdl Following_Error_Dbl;
```

```
      TD3Hdl Master_Position_Dbl;
      TD3Hdl Comp_Pos_Dbl;
      TD2Hdl DAC_i32;
      TD2Hdl Move_Time_i32;
} TD1;
// -- GLOBAL buffers for handling motor data
#define MOTOR_MAX 8
#define BUFFER_MAX 512
int32 BufferCount[MOTOR_MAX];
int32 ServoTimer[MOTOR_MAX][BUFFER_MAX];
int32 DAC[MOTOR_MAX][BUFFER_MAX];
int32 Move_Time[MOTOR_MAX][BUFFER_MAX];
float64 Commanded_Position[MOTOR_MAX][BUFFER_MAX];
float64 Position[MOTOR_MAX][BUFFER_MAX];
float64 Velocity[MOTOR_MAX][BUFFER_MAX];
float64 Following_Error[MOTOR_MAX][BUFFER_MAX];
float64 Master_Position[MOTOR_MAX][BUFFER_MAX];
float64 Compensation_Position[MOTOR_MAX][BUFFER_MAX];
// --
CIN MgErr CINRun(int32 *Device_Number_i32_0_,
int32 *Motor_Number_i32_1_8_1_,
int32 *Buffer_Length_i32_128_,
TD1 *DPR_Real_Time_Vector_Cluster,
LVBoolean *ValidData) {
      int32 DevNum = *Device_Number_i32_0_;
      int32 MNum = *Motor_Number_i32_1_8_1_ - 1;
      int BCount = BufferCount[MNum];
      int SizeL, SizeD;
      // --  Tell PMAC we're doing our thing
      PmacDPRSetHostBusyBit(DevNum, 1);

      // -- Check if PMAC is busy doing its thing
      if (!PmacDPRGetPmacBusyBit(DevNum)) {
            // -- Get Servo timer
            ServoTimer[MNum][BCount] = PmacDPRGetServoTimer(DevNum);
            /* -- If this is the first element of a new buffer - OR -
            its not the first element of a buffer - AND -
            the current servo timer value is not equal to last measurement
            THEN - fetch the DPR data into the proper buffer elements. -- */
            if ((BCount == 0) || ((BCount > 0) &&
                  (ServoTimer[MNum][BCount-1] != ServoTimer[MNum][BCount]))){
Commanded_Position[MNum][BCount] =
PmacDPRGetCommandedPos(DevNum, MNum, 1.0);
                  Position[MNum][BCount] =
PmacDPRPosition(DevNum, MNum, 1.0);

                  …
                  BufferCount[MNum]++;
            }
      }

      // -- Check for full buffer then copy to LabVIEW data structs
      if (BufferCount[MNum] == *Buffer_Length_i32_128_) {
            BCount = BufferCount[MNum];
```

```
            // -- Resize the arrays in the structure
            NumericArrayResize(iL, 1,
(UHandle *) &(DPR_Real_Time_Vector_Cluster->Servo_Timer_i32), BCount);
            (*DPR_Real_Time_Vector_Cluster->Servo_Timer_i32)->dimSize =
BCount;
…
            // --
            SizeL = BCount * sizeof(int32);
            SizeD = BCount * sizeof(float64);
            memcpy((*DPR_Real_Time_Vector_Cluster->Servo_Timer_i32)->Value,
ServoTimer[MNum], SizeL);
            memcpy((*DPR_Real_Time_Vector_Cluster->Position_Dbl)->Value,
Position[MNum], SizeD);

…
            // -- Indicate a valid buffer to caller and reset buffer counter
            BufferCount[MNum] = 0;
            *ValidData = LVTRUE;
      }
      else {
            *ValidData = LVFALSE;         // -- No valid buffer
      }
      PmacDPRSetHostBusyBit(DevNum, 0);        // PMAC can do its thing
      return noErr;
}
CIN MgErr CINLoad(RsrcFile rf)
{
      int i;
      // -- Reset the buffer counters
      for (i = 0; i < MOTOR_MAX; i++) {
            BufferCount[i] = 0;
      }
      return noErr;
}
```

The code above has the same structure as PmacDPRRealTimeMotor even though it is longer. The PMAC Busy Bit is processed as before. Instead of assigning the fetched data directly to the items of the cluster, it is stored in a set of global arrays. The BufferCount array keeps track of the next array location to be written when valid data is available for storage. There is a cryptic **IF** clause involving the ServoTimer data and BufferCount. The actual storage of fetched data in the temporary arrays requires that the ServoTimer not be the same as that of the last sample. Otherwise, the array would contain multiple samples of the data.

After the temporary arrays are updated, the CIN check the current buffer count against the desired buffer size. If there are enough samples in the temporary array, the data will be copied from the temporary arrays into the cluster of arrays passed back to the VI. This requires each array in the cluster to be resized using LabVIEW's NumericArrayResize and then the data copied from the temporary array into the data buffer that will be returned to the VI. Once the basics of this requirement are understood, clusters and arrays are easy to handle in the CIN.

There are some conditions associated with this approach. The temporary storage arrays have a fixed size defined by MOTOR_MAX and BUFFER_MAX. They have a native two-dimensional C organization. As compiled, a Motor Number larger than MOTOR_MAX cannot be used or more samples than indicated in BUFFER_MAX cannot be gathered. The CIN does not check this condition.

To resize these because of memory limitations or to get larger buffers, change these values, recompile the CIN and reload it into the CIN in PmacDPRRealTimeVectors.

It is possible to allocate these buffers dynamically using various CIN utilities. However, this introduces more complexity to the process such as allocating the buffers in the function CINLoad and deleting the buffers in CINUnLoad. For this release of PMAC Panel Pro, this approach was not utilized.

A Note About Vector CINs

To avoid unnecessary complication bullet proof PmacDPR VIs with error diagnosis, etc. have not been provided. Be aware of the fact PComm 32 Pro handles many bookkeeping issues associated with DPR. As an example, the order in which DPR operations is configured and enabled is important. If a Variable Background buffer is enabled after a Fixed Background buffer, then the Fixed Background buffer is disabled, the Variable Background buffer may move. Hence, those VIs accessing it will not return the correct data.

A similar issue arises when using the Vector VIs. Once a particular Vector buffer has been enabled for a specific number of samples, do not change the length. If this is done, unload the VIs that use the Vector VIs and reload them so that the buffer management can be reinitialized. Otherwise, it is probable that buffer bookkeeping will become garbled and strange things will happen.

A Note about Vector CIN Reentrancy

Many of the PmacDPR VIs that use CINs are reentrant. If there are two Vector VIs buffering servo data for the same motor there will be strange results. Sometimes one of the VIs will update the BufferCount and sometimes the other VI will update the count. Eventually, one of the two VIs completes the acquisition and gets the vector data leaving the other one without an acquisition. Generally this is a mistake in the application logic and can be remedied by handling the distribution of the acquired vectors in your diagram. Use one Vector VI per motor thereby guaranteeing no need for mutexs to control access to the temporary data buffers.

PmacDPRFixedBack

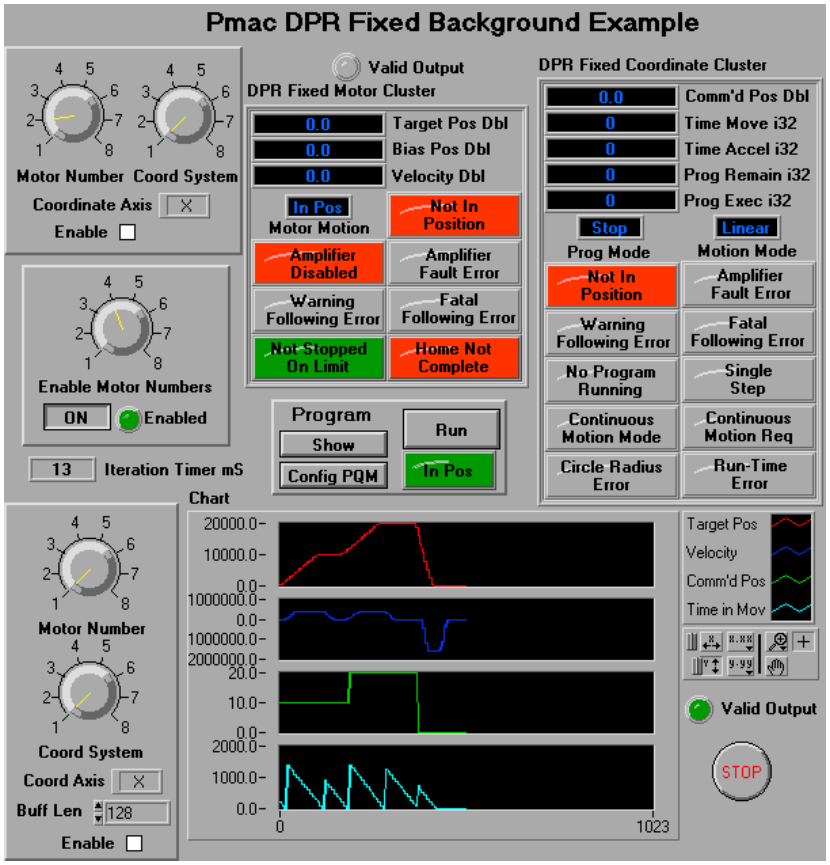
PMAC's Fixed Background data buffer mechanism automatically copies 34 selected Motor Calculation Registers, Coordinate System Control Registers, and Program Execution Registers from their native PMAC locations to DPR locations when requested. Whereas the Fixed Real Time data buffer is motor specific, the Fixed Background data is Motor and CS specific, therefore, program specific. PComm 32 Pro supplies a set of routines to read these from DPR and convert them into legitimate Intel formats. This process hides the required handshaking between PMAC and the host to avoid collisions when accessing DPR. Update of a particular item is not synchronized with a specific servo cycle.

To minimize the work of a developer and simplify the interface the PmacDPRFixedBack collection of VIs has three VIs that collect the 28 most useful items and bundle them into LabVIEW clusters. The gathering of Fixed Background data is not controlled by the servo clock therefore the data items might indeed be taken at slightly different servo times.

PmacDPRFixedBackExample

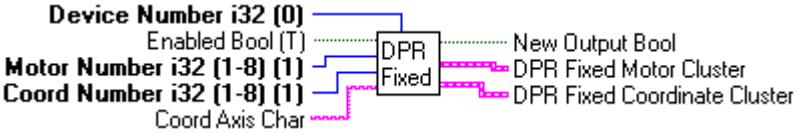
The following example demonstrates all three PmacDPRFixedBack VIs. One VI configures and enables the operation of Fixed Background data buffering, one fetches the data for a specific Motor/CS, and one buffers a set of vectors. PmacDPRFixedBack collects the data for a single motor operating in a CS. PmacDPRFixedBackVectors buffers the data exactly as PmacDPRRealTimeVectors does.

The panel for the example is shown below. The panel demonstrates the fetching of Fixed Background data for a single Motor/CS on top and the fetching of vectors for charting on the bottom. On the left are controls for selecting which motor and CS to use for the fetch and enabling the fetch from DPR. The example is different from PmacDPRRealTimeExample in that because the Fixed Background data buffer handles motor, CS, and program information, it has an encapsulated motion program that can be configured and run using the box of buttons in the middle.



To execute the example, select how many motors PMAC should copy to DPR using the **Enable Motor Numbers** knob and click the **Enable** button. Enable the operation of the display clusters by checking the **Enable** box on the top left. There are two clusters provided by PmacDPRFixedBack. A DPR Fixed Motor Cluster for the specified motor and a DPR Fixed Coordinate Cluster for the specified CS. If the **Run** button is clicked, the data in the clusters will update.

- PmacDPRFixedBack – Once DPR Fixed Background buffer operation is enabled, this VI can be used to fetch the data for a specific Motor Number and Coord Number. The input Enabled can be used to enable and disable the actual fetch. The Default, un-wired, condition is true. Coord Axis Char is a string (X, Y, Z, A, B, C, U, V, W) indicating which axis in Coord Number Comm'd Pos will represent. When New Output is true, DPR Fixed Motor Cluster and DPR Fixed Coordinate Cluster contain the most recent background data. When Enabled is false, the two output clusters contain the last valid data even though New Output is false.



Notice that there is a CS knob and a Coordinate Axis string. The values in the Coordinate cluster are for all motors in the CS, while the Commanded Pos (Comm'd Pos) is for a specific Axis in the CS. In the example the X axis is specified. If the axis is changed to Y and **Run** is clicked again, notice that the Comm'd Pos does not update because the Y axis is not defined in CS 1.

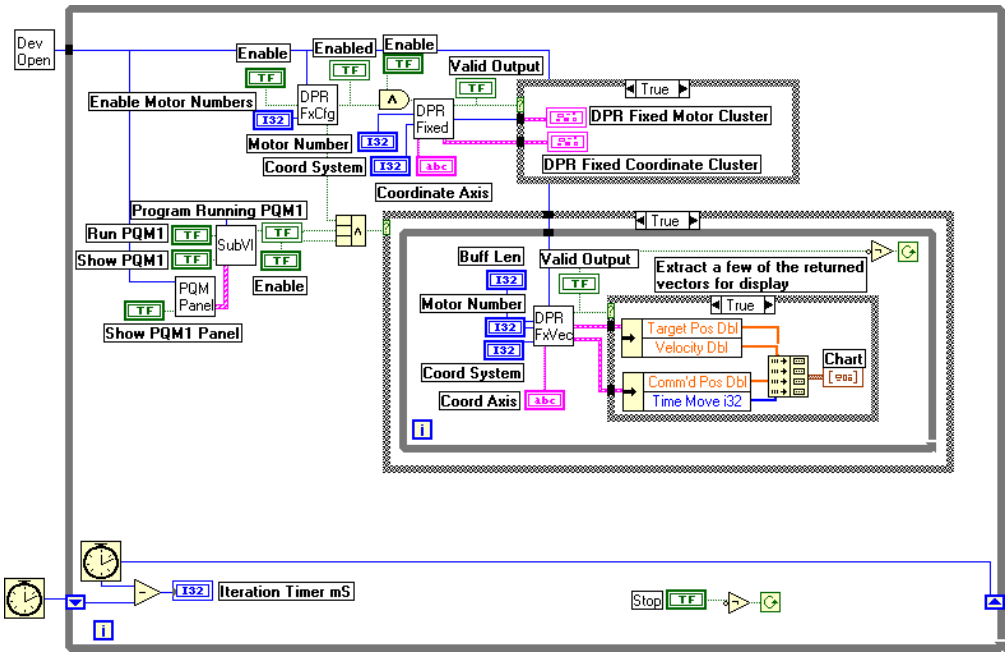
The operation of the example is a little different when the Vector operation is enabled by checking the **Enable** box. Nothing happens until the program is run. This is due to the organization of the example not the associated Vector VI. When the Vector portion of the example is enabled and the **Run** button is clicked, the chart will begin updating. Disable the top portion so that it does not steal samples from the Vector operation.

Notice that the displayed data is more quantized because of the DPR data is updated in the background. Fixed Background data is program related and therefore computed at a slower rate and updated only when requested. The other thing to note is that the velocity for Fixed Background operation is in encoder counts per minute, whereas Real Time motor velocity is in scaled Ix09 counts per servo cycle.

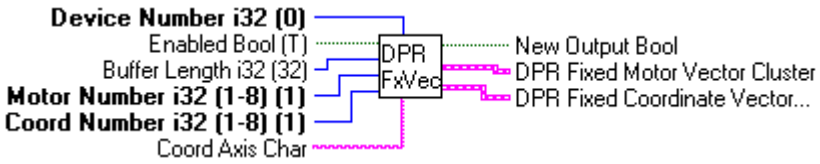
The diagram for the example shows three VIs. One VI configures the operation, one fetches the data for a single motor and one fetches the vector data. In addition, there is a small diagram to handle the encapsulated motion program. A **CASE** structure is used to control the update of the display clusters. The PmacDPRFixedBackConfig and PmacDPRFixedBack VI operate similarly to their PmacDPRRealTime versions in that the Enable terminals operate the same way.

- PmacDPRFixedBackConfig – DPR Fixed Background buffer is configured to update motor information for all motor between one and Enable Motor Numbers when Enable is true. Operation is disabled when Enable is false. Output Enable is true when operation is enabled. The VI maintains the state. Operation of DPR Fixed Background buffers overlaps with DPR Real Time operation in that the number of motors enabled must be the same.

Note that the PmacDPRFixedBack vector VI is wrapped in a **CASE** structure that is executed only when the configuration VI is enabled, the program is executing, and the proper **Enable** button on the panel is checked. Hence, when the program is started, the gathering of the vectors can begin. The samples are accumulated as fast as possible because the actual Vector VI is buried in a **WHILE** structure that executes until the entire vector is accumulated and then the rest of the system gets a chance to run. This structure is not required but demonstrates another way to organize a gather.



- PmacDPRFixedBackVectors – Once DPR Fixed Background buffer operation is enabled, this VI can be used to fetch the data for a specific Motor Number and Coord Number. The input Enabled can be used to enable and disable the actual fetch. The Default, un-wired, condition is true. Coord Axis Char is a string (X, Y, Z, A, B, C, U, V, W) indicating which axis in Coord Number Comm'd Pos will represent. When New Output is true, DPR Fixed Motor Cluster and DPR Fixed Coordinate Cluster contain the most recent background data. When Enabled is false, the two output clusters contain the last valid data even though New Output is false.



The details of the CINs associated with each VI will not be covered here. They are similar to those presented in the PmacDPRRealTime collection. Specific details are contained in source code comments.

PmacDPRNumeric

PMAC's architecture permits access to any unused DPR memory for whatever purposes using M-Variables. These may be scratch registers or, in the case of DPR, registers through which data can be passed between the host and PMAC. By avoiding the translation involved in standard ACSII communication data can be moved between the host and PMAC more quickly and it does not utilize precious P or Q-Variables. The process requires no explicit handshaking between PMAC and the host to avoid collisions when accessing DPR.

The PmacDPRNumeric collection consists of a number of VIs that uses Call Library Function VIs to access DPR. The simplicity of the interface does not require CINs. We show how to access individual memory locations as doubles, long (i32) and short (i16) integers, bit fields, and booleans. The example uses encapsulated PLCs and motion programs to generate and process the data transferred between the example PMAC Panel Pro applications and PMAC. If DPR is used for this purpose, the examples are useful.

DPR Addresses and Data Organization

The mapping of memory addresses between the host computer on one side and PMAC's address space on the other side is simple. To PMAC, DPRAM appears as standard memory in the range $D000 - $DFFF, which can be thought of as 4K of long (48-bit) words or 8K of single (24-bit) X/Y words. This memory is accessed using M-Variables mapped to this address range. Depending on the DPR mechanisms used (Real Time, Fixed Background, Variable Background, etc.), the lower portions of this memory space are automatically allocated. Hence, anything above the end of this space up to $DFFF can be used for any purpose. There is no easy way to allocate addresses for DPR Numeric access automatically until all other automatic features have been allocated. In the next section on DPR Variable Background buffers, it is outlined how to determine the end of this allocated space. For now, use DPR memory between &DE00 and &DFFF.

To the host computer, DPR appears as 8K 16-bit words of memory. Each 24 bit PMAC X or Y word thus takes two 16-bit memory addresses. A PMAC long or float (48-bits) takes four memory addresses. Fortunately, PComm 32 Pro handles the host computer memory mapping and PMAC handles the required bit and byte manipulations to map Motorola 56K data formats to Intel data formats.

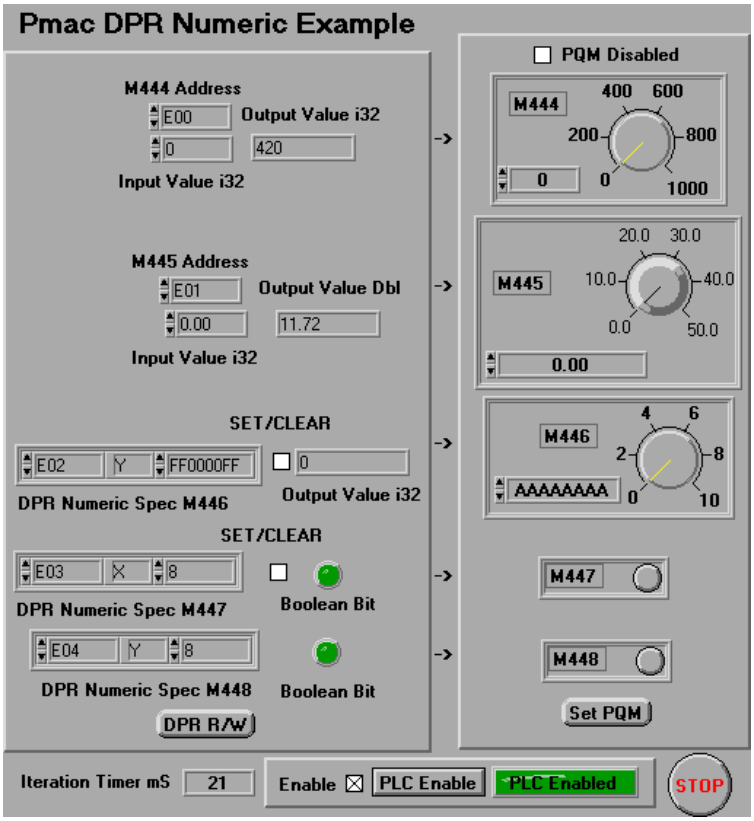
PmacDPRNumericExample

The following example demonstrates the use of DPR for communicating numeric data between the host and PMAC. PMAC executes a PLC that generates and responds to the register data accessed using M-Variables that are mapped to DPR. The data is also accessed using the PQM collection of VIs to demonstrate the differences in access speed and bypass the mechanisms that field access introduces.

Unlike the previous PmacDPR examples that required a configuration step prior to accessing the data, PmacDPRNumeric requires no configuration other than assignment of memory addresses.

The panel for the example is shown below. On the left is a box containing controls/indicators that access M444 – M448 using PmacDPRNumeric VIs. On the right are several controls that access the same data using the ASCII PQM collection of VIs. Each of the M-Variables can be accessed using either method. From the top down are M444 an integer, M445 a double, M446 a bit field in a Y address, M447a single bit in an X address, and M448, another bit in a Y address. Access to the M-Variable data using DPR requires an address, bit number, or field specifier. These are also shown on the left portion of the panel.

At the very bottom are a few controls to enable and monitor an encapsulated PLC that generates M-Variable data that is available to the host and PMAC simultaneously because it exists in DPR. Check the **Enable** box and then click the PLC Enable button and the PLC program will begin executing. The indicators for each of the M-Variables will begin updating immediately with the data being generated in PMAC's PLC. If the PQM Disabled box on the right is checked, these indicators will begin updating also with a noticeable increase in the interval timer. This is because of the large overhead required to process the required ASCII commands.



M-Variables and VI Address Specification

Before getting into the example deeper, look at the PLC M-Variable definitions shown below. These specify the addresses where PMAC will place the data during its writes to M-Variables and fetch the data when it reads an M-Variable. The address modifier DP defines a 32-bit long integer in DPR handled as the lower 16-bits of both X and Y addresses. The F modifier defines a 32 bit floating point value in DPR also handled as the lower 16 bits of both X and Y addresses. PMAC firmware and PComm 32 Pro handle the required bit and byte manipulations to convert the raw representation into Intel and Motorola formats. M447 and M448 are single bits defined in simple 24-bit X/Y words.

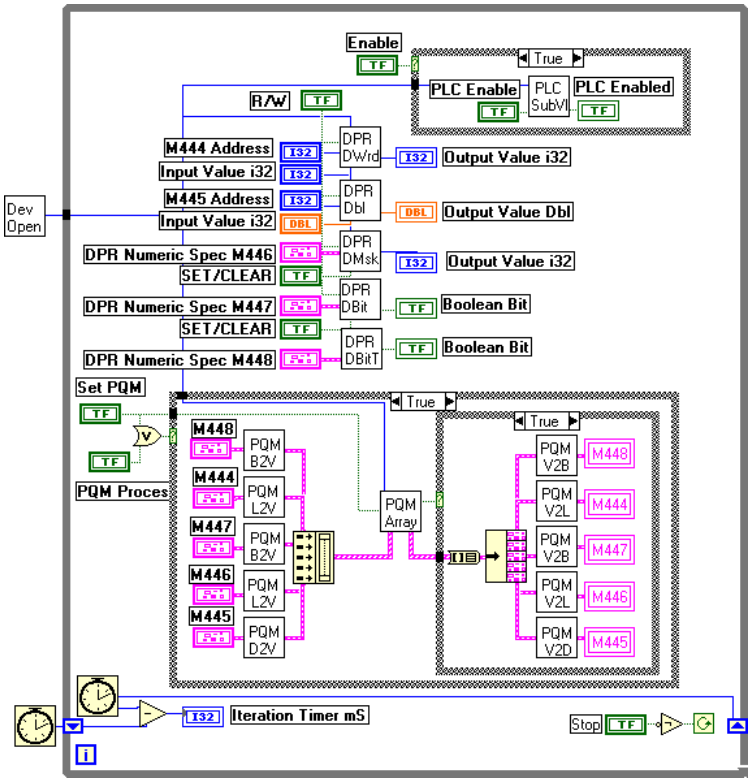
```
M444->DP:$DE00
M445->F:$DE01
M446->DP:$DE02
M447->X:$DE03,8,1
M448->Y:$DE04,8,1
```

To access these variables with the PmacDPRNumeric collection of Vis, a truncated version of the memory address is required. A PMAC M-Variable defined at $DE45 become 0xE45 to PMAC Panel Pro. PComm 32 Pro handles the absolute memory mapping while the PmacDPRNumeric VIs compute the address offset required by PComm 32 Pro. For M-Variables defined as F and DP, nothing more is required. For M-Variables defined as X, Y, or specific bit fields, a cluster defining the base address, modifier, and field or bit number is required.

When looking at the panel, only the address is required for M444 and M445. M446, M447, and M448 require the cluster. These are covered in detail later.

The diagram for the example has a section at the bottom for handling the PQM controls and a case statement at the top for handling the encapsulated PLC. The PLC encapsulation VI is wrapped inside the case statement so that it can be disabled and the impact of its execution on timing can be seen. The PQM approach can be used to validate the results of bit field manipulation that are masked by the DPR mechanism.

The five VIs in the middle handle the transfer of data between the host and PMAC using the same read/write architecture used for PmacIVar, PmacMemory, etc. All that is required is the address for DWord or Double memory or a DPR Numeric Spec cluster for field and bit access.

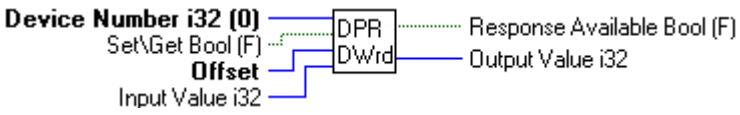


PmacDPRNumeric VIs are discussed in two groups. The first group covers word, double word, and floating point DPR M-Variables. The second group covers the bit and bit-field VIs. In a later example, an approach for grouping collections of DPR M-Variables into a cluster that can be handled using a CIN will be demonstrated.

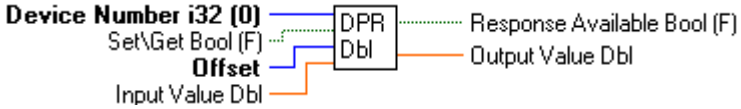
- PmacDPRNumericDWord – This VI is used to set or get PMAC long M-Variables defined in DPR as M447->DP:$DE03. PMAC handles the translation of PMAC's representation into Intel format when the DP specification is used.

When Set/Get is false (default state), Response Available is true and Output Value is the value in DPR located at Offset. When Set/Get is true, Output Value = Input Value, Response Available is false and the specified DPR location is set.

The mapping of PMAC addresses to PC addresses is involved and requires a bit of work to understand. In general, each 24-bit PMAC word requires one 32-bit PC word. For PMAC specified as M447->DP:$DE03, the offset should be $E03.

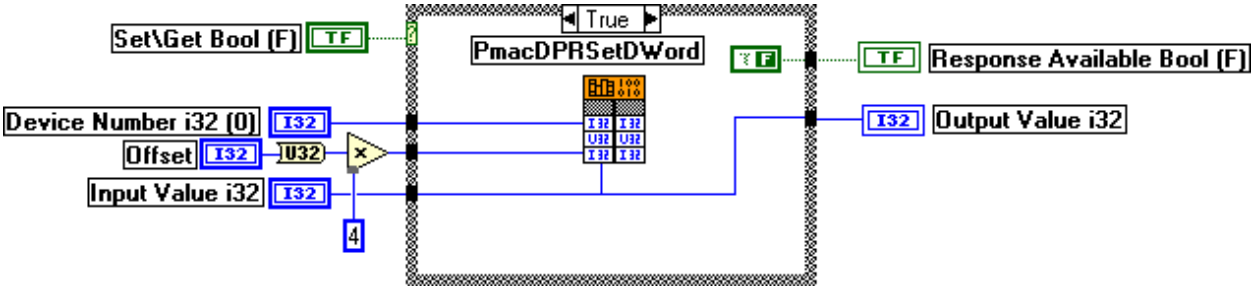


- PmacDPRNumericDbl – This VI is used to Set or Get PMAC double M-Variables defined in DPR as M447->F:$DE03. PMAC handles the translation of PMAC's representation into Intel format when the F specification is used.



- PmacDPRNumericWord – This VI is used to set or get PMAC long M-Variables defined in DPR as M447->DP:$DE03 where the equivalent intel representation is 16 bits. PMAC handles the translation of PMAC's representation into Intel format when the DP specification is used.

The implementation of these VIs closely follows that used by PmacMemory and PmacIVar. The VI will get the specified value unless the Set\Get input is true in which case it does a set operation. The diagram shown below demonstrates how this is done with a Call Library VI using the PmacDPRSetDWord function in PComm 32 Pro. The false case (Get operation) uses the PmacDPRGetDWord function. The offset supplied by the diagram is multiplied by 4 to get the actual memory offset of the M-Variable in DPR as seen from the host.



DPR Bits and Bit Fields

The three VIs presented above require only the offset to determine the address of the desired data. When accessing bits and bit fields the information contained in the PmacDPRNumericSpec cluster is required.

DPR Numeric Spec Cluster A cluster of items required to describe a DPR mapped PMAC M-Variable for bit and field access.

Address i32 Hexadecimal integer specifying DPR address offset. For example, PMAC Addresses such as:

```
M445->F:$DE01
M446->DP:$DE02
M447->X:$DE03,8,1
```

Become

E01, E02, and E03 respectively (M446->DP:$DE02)

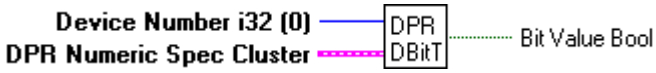
X/Y String A single character string (X or Y) defining the type of data. Not for L or DP.

Mask/Bit i32 A hexadecimal value used to define a bit number for single bit operations or a multi digit hexadecimal number defining a mask for multi-bit operations.

The VI PmacDPRNumericSpec is embedded in the bit and bit field VIs covered next, and converts the address specification into an actual DPR memory offset. Look at the actual diagram for this VI to understand how this is done. Generally, although these are named as DWord operators, the individual bits are defined in a 24-bit X/Y word.

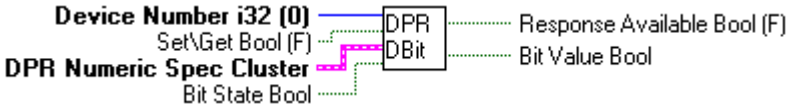
The bit VIs are:

- PmacDPRNumericDWordBitTest – This VI queries the DPR DWord bit specified by DPR Numeric Spec Cluster and returns the value in Bit Value.



- PmacDPRNumericDWordBit – This VI operates on the DPR DWord bit specified by DPR Numeric Spec Cluster.

  When Set/Get is false (the default state), the value of the bit is queried and returned by Bit Value with Response Available true. When Set/Get is true, the specified bit is set to the value of Bit State - either true or false.

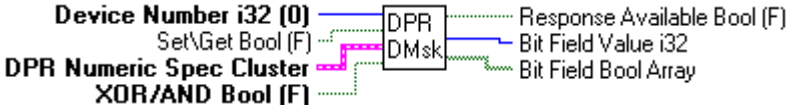


  Bit field operations are a little more complex. The following VI allows the specification of an entire X/Y word and sets or clears multiple bits in a single operation depending on the control input.

- PmacDPRNumericDWordSetMask – This VI operates on the DPR DWord bit field specified by DPR Numeric Spec Cluster.

  When Set/Get is false (the default state), the Mask specified by DPR Numeric Spec Cluster is AND'd with the specified address to produce the output Bit Field Value. Response Available is true.

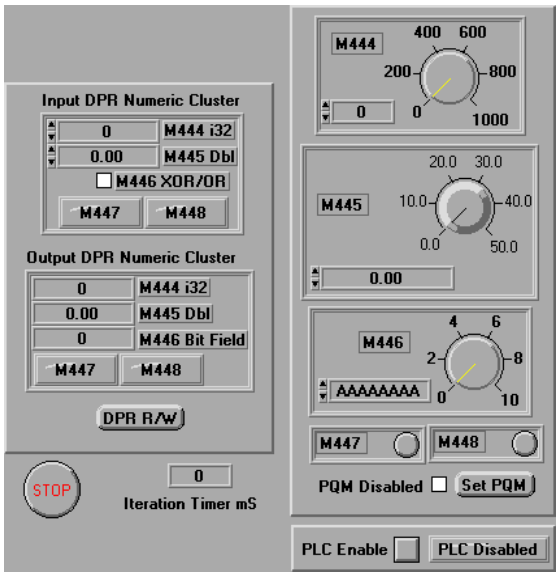
  When Set/Get is true, the Mask is either OR'd or XOR'd with the contents of the field at the specified address. If XOR/OR is false, the mask is OR'd with the contents of the field at the specified address thereby setting bits specified by the mask. If XOR/OR is true, the mask is XOR'd with the contents of the field at the specified address thereby clearing the bits specified in the mask.



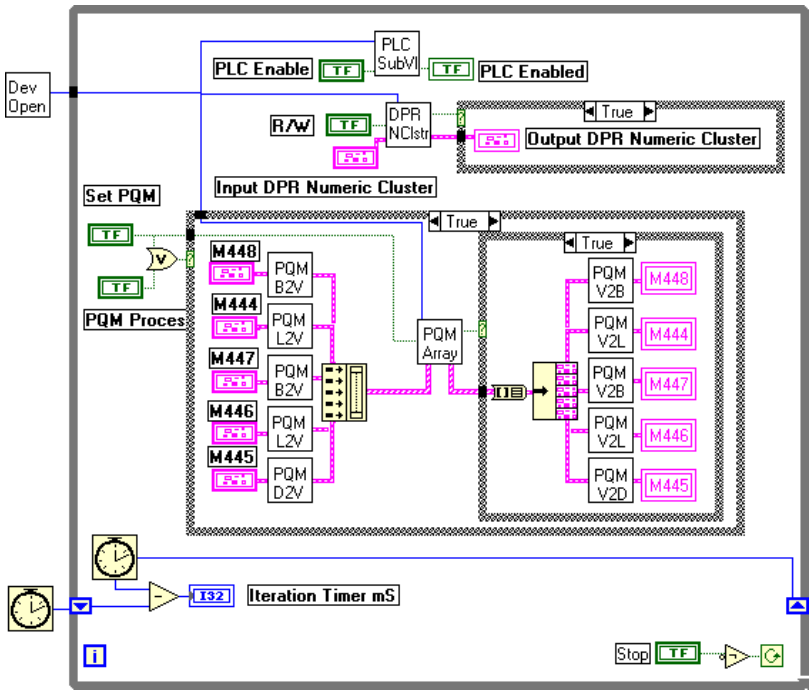
PmacDPRNumericClusterExample

This example extends the previous example by defining a cluster containing a set of DPR numeric data. The purpose of doing this is to hide the addresses inside the VI and get ready for another example that will use a CIN to access the DPR numeric data.

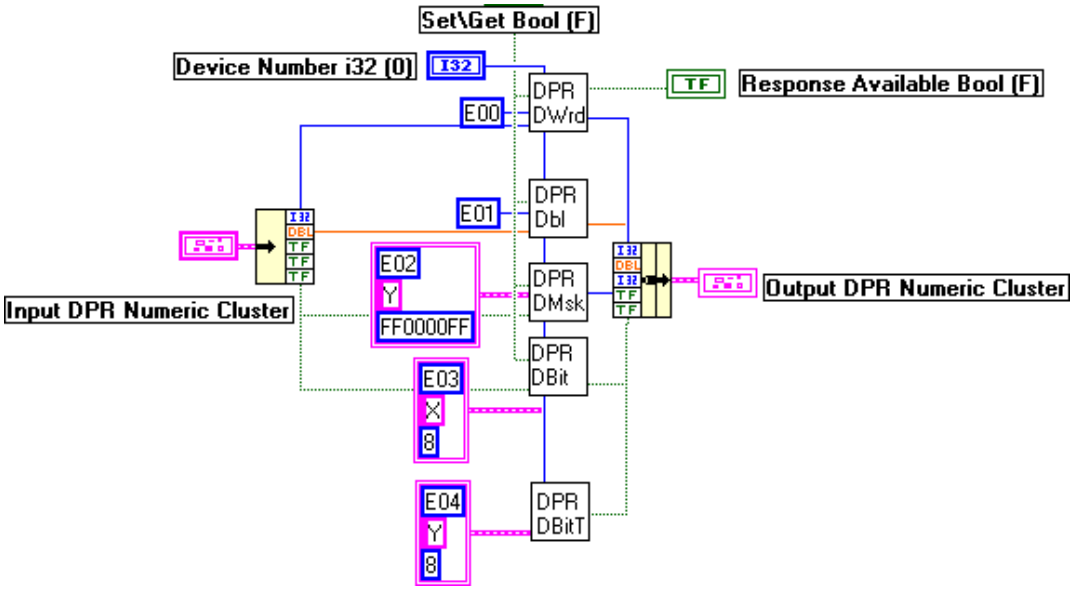
The example operates the same as the previous one except that a large number of controls for defining the M-Variables have been reduced to a single cluster as shown in the panel.



The handling of the five VIs in the earlier example diagram is reduced to a single VI.



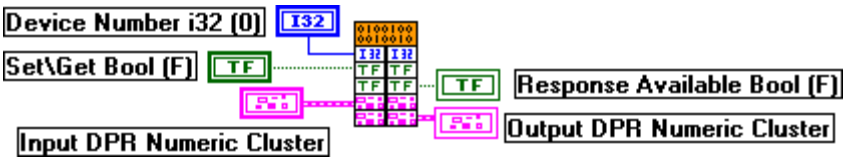
The diagram for the DPR Numeric Cluster VI is shown below. Copy this and modify it to support individual requirements. Note that the addresses are specified by cluster constants. These can be created using the right mouse button and selecting **Create Constant**.



PmacDPRNumericCINClusterExample

This example takes the previous example a little further and implements the actual handling of DPR data using a CIN. This can be useful if there is many data being transferred, or there is trouble maintaining dozens of PmacDPRNumeric Vis, or have special data handling requirements that benefit from a CIN.

The diagram handling the five M-Variables in the example above are reduced to a single CIN VI that will handle the reading and writing of the data with direct PComm 32 Pro calls. To simplify development of these CINs, PmacDPR defines a set of macros that make life easy.



The code for the CIN is shown here. To use the macros, include the file PmacDPRNumericCINCluster.h located in PmacDPR.

```
#include "extcode.h"
// -- PmacDPRNumeric Macros --
#include “PmacDPRNumericCINCluster.h"
#include <pmacu.h>
#include <dprrealt.h>
/* typedefs */
typedef struct {
      int32 M444_i32;
      float64 M445_Dbl;
      int32 M446_Mask_i32;
      LVBoolean M446_XOR_OR;
      LVBoolean M447_Bit;
      LVBoolean M448_Bit;
} TD1;
typedef struct {
      int32 M444_i32;
      float64 M445_Dbl;
      int32 M446_Bit_Field;
      LVBoolean M447_Bit;
      LVBoolean M448_Bit;
} TD2;
CIN MgErr CINRun(int32 *Device_Number_i32_0_, LVBoolean
*Set_Get_Bool_F_,
LVBoolean *Response_Available_Bool_F_,
TD1 *Input_DPR_Numeric_Cluster,
TD2 *Output_DPR_Numeric_Cluster) {
      int32 DevNum = *Device_Number_i32_0_;
      // -- Using the macros
      PmacDPRNumericDWord (*Set_Get_Bool_F_,
0xE00,
Input_DPR_Numeric_Cluster->M444_i32,
Output_DPR_Numeric_Cluster->M444_i32);
PmacDPRNumericDouble(*Set_Get_Bool_F_,
0xE01,
Input_DPR_Numeric_Cluster->M445_Dbl,
Output_DPR_Numeric_Cluster->M445_Dbl);
PmacDPRNumericDWordMask(*Set_Get_Bool_F_,
0xE02,
Input_DPR_Numeric_Cluster->M446_Mask_i32,
Input_DPR_Numeric_Cluster->M446_XOR_OR,
Output_DPR_Numeric_Cluster->M446_Bit_Field);
PmacDPRNumericDWordBit (*Set_Get_Bool_F_,
0xE03, 'X', 8,
Input_DPR_Numeric_Cluster->M447_Bit,
Output_DPR_Numeric_Cluster->M447_Bit);
PmacDPRNumericDWordBitTest (0xE04, 'Y', 8,
Output_DPR_Numeric_Cluster->M448_Bit);
```

```
      PmacDPRNumericResponse(*Response_Available_Bool_F_,
*Set_Get_Bool_F_);
      // --
      return noErr;
}
```

As with all CIN nodes, LabVIEW writes the function declaration and defines the parameter types. The macros require that the device number must be defined DevNum no matter what it is in the parameter list. The macros perform a conditional test for read/write operations, address calculations, and PComm 32 Pro operations. Generally, the macros require an address and a pointer to the input and output elements of the cluster. Bit operations require the bit number and Mask or Bit Field operations require the mask, field mask, and XOR/OR operator. The actual C-code for the example contains the macros and their actual C-counterparts to illustrate the operations performed.

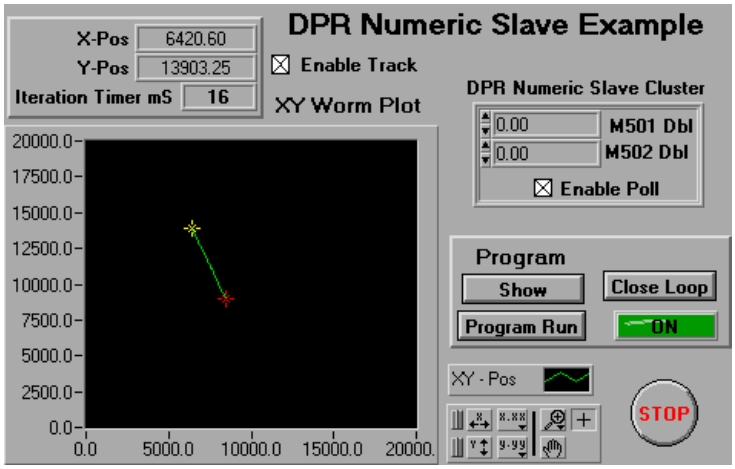
PmacDPRNumericSlaveExample

This example uses PmacDPRNumeric and PmacDPRRealTime capabilities to build an application that allows moving a 2-axis X-Y table with the mouse. To accomplish this, the motion program shown below uses DPR mapped M501 and M502 to define the target position for motors 3 and 4. M500 is a Boolean used to control a loop that breaks the target position into a set of smaller moves. The program is encapsulated in a VI for easy use.

```
; USE CS &3
&3
#3->10x
#4->10y
; -- These are DPRAM mapped target coordinates
M501->F:$DE06    ; X coordinate
M502->F:$DE07    ; Y coordinate
; -- RUN BOOL
M500->Y:$DE05,8,1
; --
open prog 61 clear
;hm1..4
i13 = 100
P209 = 1000    ; Vector distance per increment
P211 = 10000    ; Starting positions
M501 = 10000
P212 = 10000
M502 = 10000
tm10
; -- Move tracking
while (M500=1)

    P221 = M501 - P211    ; Cartesian distance to go
    P222 = M502 - P212

    P200 = sqrt(P221 * P221 + P222 * P222)
    if (P200 > P209)      ; If longer than increment
        P200 = P209 / P200    ; Fraction of distance
        P211 = P211 + P221 * P200
        P212 = P212 + P222 * P200
    else
        P211 = M501
        P212 = M502
    endif
```

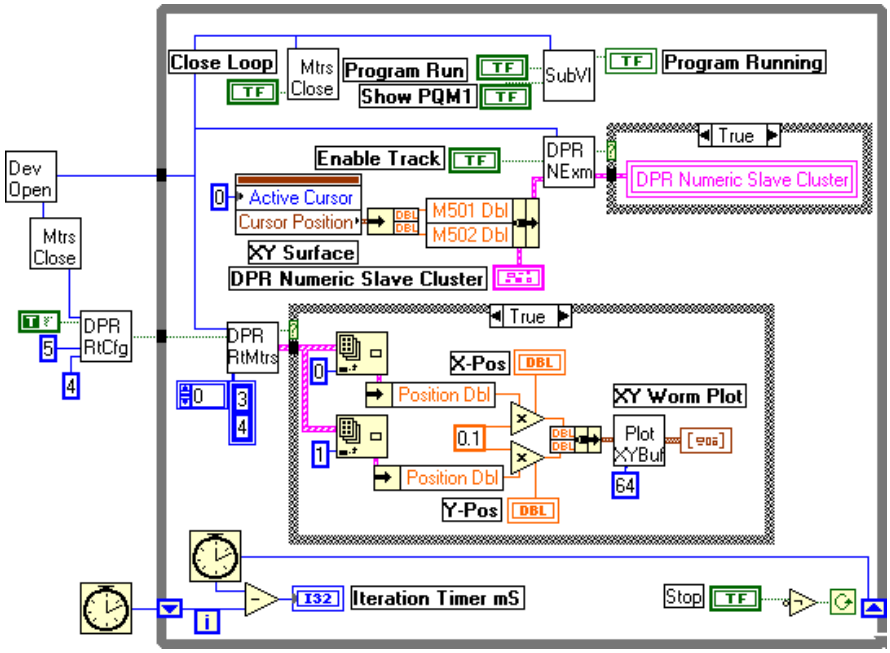
```
    x(P211)  y(P212)
endwhile
close
```

The panel for the example is shown below. When the VI is running and **Enable Track** is checked, the values in DPR Numeric Slave Cluster - M500, M501, and M502 - are written to memory continuously for use by the motion program. Checking **Enable Poll** sets M500 true so that clicking the **Program Run** button enables the program loop. If the box is not checked prior to starting the motion program, the loop in the motion program will not execute. The **Close Loop** button is provided to close the servo loops if they are not closed, or the initial rapid move to home results in a fatal following error.



The indicators on the middle left of the panel display the actual X/Y motor positions as retrieved by PmacDPRRealTimeMotors. The yellow cursor in the plot is the target position for the move and determines the values of M501 and M502. When the yellow cursor is clicked and dragged to a new position, the target position for the move is set and executed by the motion program. The green worm will begin moving toward the yellow cursor with the red cursor bringing up the rear.

The diagram for the example is similar to those already discussed. The motion program wrapper VI on top handles the program execution. Below this is the VI handling the DPR Numeric cluster containing M500-M502.



The query for motor positions used to update the plot is provided by PmacDPRRealTimeMotors. The configuration of this capability is done outside the execution loop. When the motor positions are fetched, array is indexed, the position is unbundled, converted to CS units (multiplied by 0.1), used to update the panel indicators, and bundled into and X-Y point for the X-Y Chart buffer. Remember that PmacDPRRealTimeMotors does not perform CS conversions. In addition, the PmacDPRFixedBack VIs can be used to obtain the motor position in CS units.

To handle target position generation for the move, retrieve the position of Cursor 0 from a chart attribute node and update the values in the DPR Numeric Slave Cluster. These values will be written by PmacDPRNumericSlaveCluster as long as Enable Track is true.

PmacDPRVarBack

PMAC's Variable Background data buffer mechanism allows the specification of 128 memory addresses to copy from the PMAC locations to DPR locations when requested. Whereas the Fixed Background data buffer allows access to predefined registers and locations, the Variable Background mechanism allows access to anything. PComm 32 Pro supplies a set of routines to read the copied data from DPR and convert them into legitimate Intel formats. This process hides the required handshaking between PMAC and the host to avoid collisions when accessing DPR. PComm 32 Pro supports a limited ability to write to DPR from the host and copy this data to its native location. PmacDPRVarBack provides hooks for this interface but does not currently implement this capability.

To minimize work as a developer and simplify the interface, the PmacDPRVarBack collection of VIs provides three VIs. One configures one or more Variable Background buffer, one fetches its contents, and one buffers the data into vectors. To aid in specifying items, the configuration VI uses the PmacGatherSpec cluster that forms the heart of the PmacAddress and PmacGather collections.

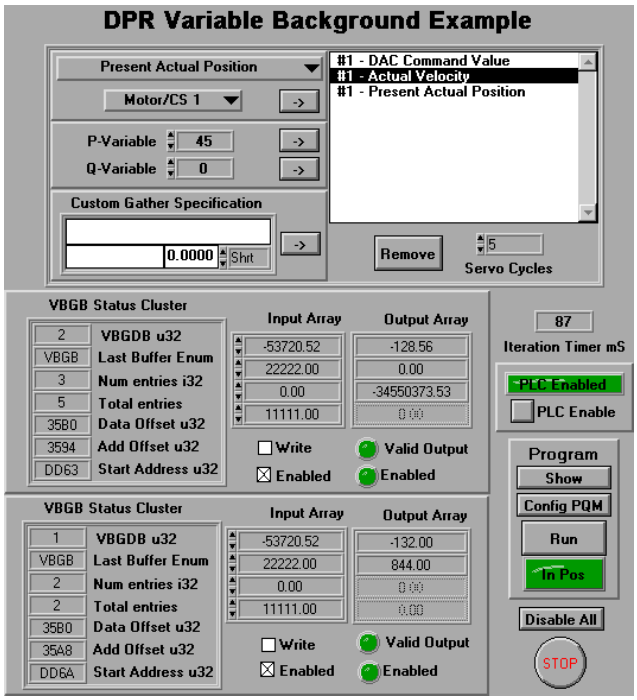
The gathering of Variable Background data is not controlled by the servo clock, therefore the data items might be taken at slightly different servo times.

PMAC's Variable Background mechanism is very sophisticated and an integral part of PMAC. Some specific issues will be covered in the next section.

PmacDPRVarBackExample

The following example demonstrates the configuration of two Variable Background data buffers. This allows building of buffers that support specific requirements. For instance, one buffer can be declared for each motor and coordinate system in the system. Then another one can be declared for each I/O device, and one to monitor a collection of miscellaneous items. Gather some of them as vectors, some for indicator clusters, and some for background computations. The only limitation is that more than 128 items between all of them cannot be declared.

The panel for the example, shown below, allows the definition of two independent buffers. The support clusters, etc. are collected into boxes in the bottom left quadrant. Each buffer has a VBGB Status Cluster containing information about the individual buffer, its location in DPR, and the entire pool of buffers. To the right of this are an Input Array and an Output Array. The Output Array contains the data for the specified buffer. The Input Array is provided but, as noted, not supported. Below these items is a Write check box (not supported), an Enabled check box, and two indicators. On the far right are a few collections of buttons to control the associated PLC and motion program.



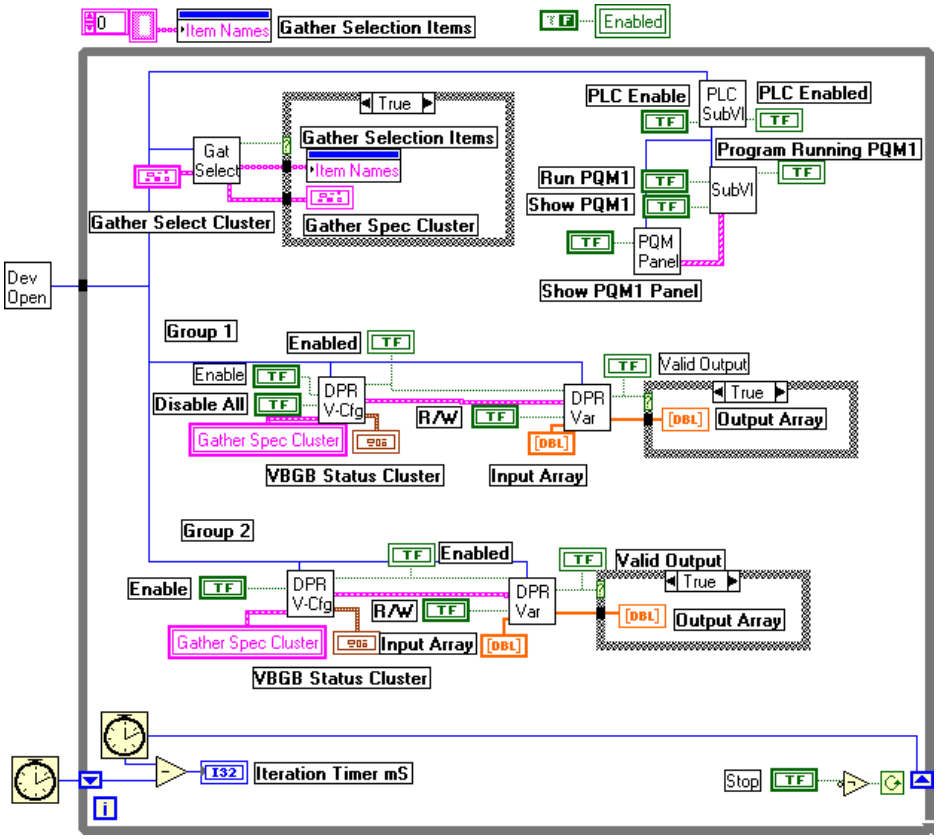
To run the example, specify a set of items to gather. Use the PmacGatherSelect cluster on the top left. The operation of this is outlined in the section on PmacTerminalGather. As shown in the example, there are three items defined. To create a Variable Background buffer, select the items to be in the buffer and check the **Enabled** box. In the example, the upper set was created to handle these three items. The lower set was created to handle P44 and P45 that are generated by the PLC program.

The VBGB Status Cluster maintains information about the individual buffers and the entire pool. Refer the VI Reference for details of each item. The top status cluster item VBGDB indicates that it is buffer 2, whereas the bottom cluster indicates it is buffer 1. Buffers are assigned in the order created. Num Entries indicates that the top buffer has three items (as defined by the Gather Spec) and the bottom buffer has two items. When the first buffer was created, the Total number of entries was 2. After the second buffer is created, the Total number of entries is 5. Start Address indicates the start of the buffer in DPR. This is the last location reserved by DPR. The Start Address of the first buffer created indicates the first address that can be used for DPR Numeric access.

Notes on the Use of PmacDPRVarBack

Variable Background buffers should be created after all other DPR mechanisms have been enabled. PComm 32 Pro might and sometimes does move things around when reconfiguring DPR. If more than one buffer is created, do not delete a previously created buffer. Again, PComm 32 Pro will shift things around and it is likely that the remaining buffers will contain garbage. If a buffer is deleted, delete and recreate the remaining buffers. If the **Enabled** box is un-checked, that buffer is deleted. Chances are that the remaining buffer will give out garbage data. Un-check the remaining buffers and then re-check them. Things will now behave as expected.

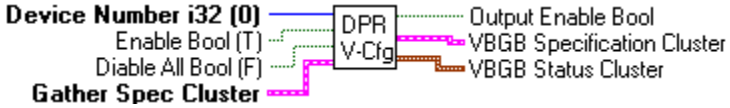
The diagram for this example demonstrates how easy it is to create and access a buffer. At the top left is the VI to handle the Gather Select Cluster. Review the PmacTerminalGather section to answer any questions. On the top right are the PLC and motion program handlers.



There are two almost identical configurations to handle the two buffers. Each consists of a configuration VI PmacDPRVarBackConfig and PmacDPRVarBack to actually fetch the data. The configuration VI requires a Gather Spec Cluster and produces a VBGB Specification Cluster for the handler. The VBGB Status Cluster is not required by other VIs but serves a useful diagnostic purpose.

- PmacDPRVarBackConfig – This VI creates a set of Address Items specified by Gather Spec Cluster using the DPR Variable Background when Enable is true. The VI produces a VBGB Status Cluster with relevant information about this buffer and a VBGB Specification Cluster containing information required to actually fetch the data using PmacDPRVarBack and PmacDPRVarBackVectors.

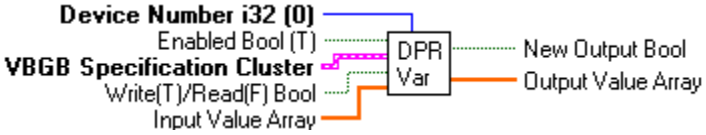
Operation is disabled when Enable is false. Output Enable is true when operation is enabled. The VI maintains the state. This VI can be used multiple times to create sets of VBGB Address Items. See the documentation for limitations on how many sets can be created and their size.



Variable Background buffers allow the gathering of the contents of any memory location, X, Y, DP, etc. To handle all data types the data fetched from DPR is all treated as a double. This allows PmacDPRVarBack to treat the fetched items as an array rather than have VIs to handle each type or implement a complex typing mechanism. To access bits from a particular item, index the array, convert it to an integer, and use it. The Input Array and R/W inputs are not supported yet.

- PmacDPRVarBack - If DPR Variable Background buffer operation is enabled, this VI can be used to fetch the data specified during the configuration. The input Enabled can be used to enable and disable the actual fetch. The Default, un-wired, condition is true. When New Output is true, Output Value Array contains the most recent background data. When Enabled is false, Output Value Array contains the last valid data even though New Output is false.

The Write/Read and Input Value Array inputs are not functional currently. Future releases may implement this capability.

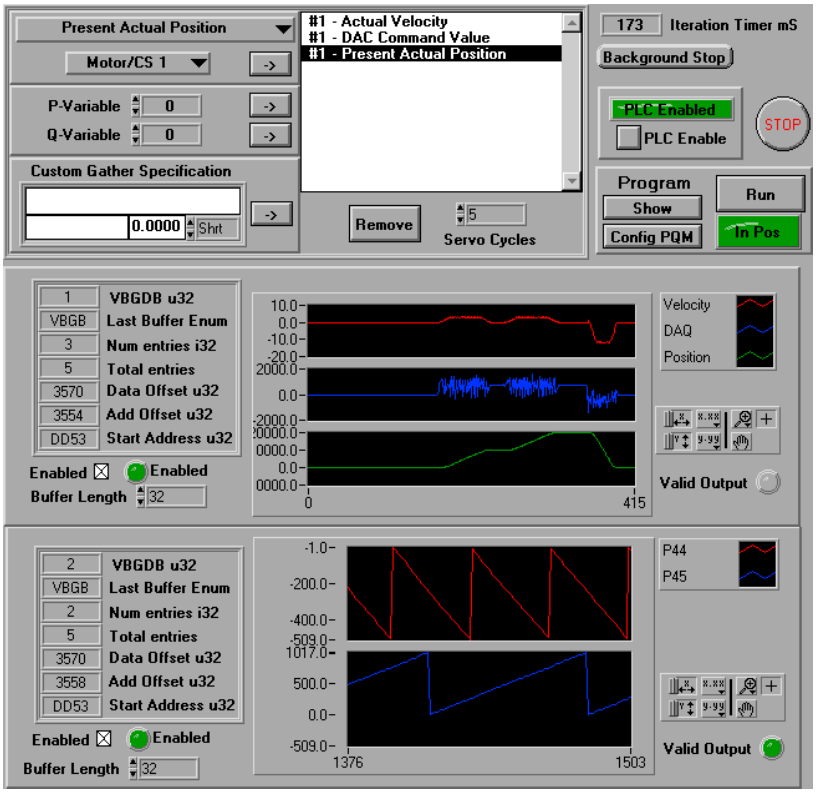


Note on Supporting PmacDPRVarBack CINs

The CINs to support this collection of VIs are significantly more involved than the previous collections. The comments in the code provide enough information to try to tackle changes alone.

PmacDPRVarBackVectorExample

This example replaces PmacDPRVarBack with PmacDPRVarBackVectors. Its use is almost identical to PmacDPRVarBackExample. The only difference is that the bottom buffer has a PmacGatherSpec cluster defined as constant containing entries for P44 and P45. Therefore, the Address Items need not be specified. Simply check the **Enabled** box. The diagram is not presented for the same reasons.

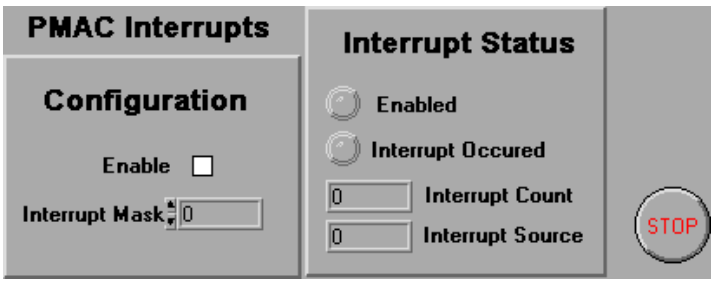


INTERRUPTS

This section documents an emerging interface to PMAC's interrupt system. The information presented here is preliminary and not supported yet.

PmacInterruptExamp

To run this example, check the **Enable** box. Interrupt Mask defines which sources are enabled. The default value of zero enables all sources. The In Position flag generates an interrupt and is a good test. When an interrupt occurs the Interrupt Occurred LED is on and the count and source update.



Always disable interrupts when the application halts.

- PmacInterruptConfig – When Enable is true, enable interrupts. Indicate the availability of a handler by the output Enabled. When false, disable interrupts. Always disable interrupts when you application is not executing.

Interrupt Occurred is true whenever this VI checks the handler and determines an interrupt has occurred. Interrupt Count indicates the number of interrupts since the last service. Interrupt Source specifies which source. See the PMAC User Manuals for details.